

Министерство науки и высшего образования Российской Федерации

Курский государственный университет

Кафедра программного обеспечения и администрирования
информационных систем

Основы функционирования ЭВМ

Лабораторный практикум (Выпуск 3)

Курск 2021

Основы функционирования ЭВМ: методические указания к выполнению цикла лабораторных работ (Выпуск 3, перераб. и доп.) / сост. А. П. Жмакин; Курск. гос. ун-т. – Курск, 2021, 65 с.

Данное пособие можно рассматривать как продолжение (или альтернативу) методических указаний [2] и предназначено прежде всего для знакомства с принципами взаимодействия процессора с внешними устройствами ЭВМ.

В отличие от [2], в настоящем пособии используется программная модель учебной ЭВМ, названная нами fN8 – модель 8-разрядной двоичной ЭВМ с архитектурой фон Неймана. В состав модели, помимо процессора и памяти, включены внешние устройства (контроллер клавиатуры, символьный и графический дисплей, таймерные системы, контроллер семисегментной индикации и матричной клавиатуры), причём открытая архитектура и заложенный в модель резерв адресного пространства ввода/вывода позволяет расширять набор подключаемых ВУ.

В выпуске 3 отражены изменения, внесённые в программную модель fN8 и добавлены указания к выполнению двух лабораторных работ (2.1 и 2.2)

Пособие предназначено для студентов направления 02.03.03 *Математическое обеспечение и администрирование информационных систем* (курс «Архитектура вычислительных систем»), а также может быть использовано студентами направлений 09.03.01 и 09.04.01 *Информатика и вычислительная техника*, 01.04.02 *Прикладная математика и информатика* в курсе «Архитектура компьютеров».

Содержание

I	Описание модели ЭВМ	6
1.	Архитектура учебной ЭВМ fN8	6
1.1.	Основные характеристики модели	6
1.2.	Организация адресного пространства	6
1.3.	Директивы компилятора	8
1.4.	Ресурсы процессора	9
1.5.	Подсистема прерываний fN8	10
2.	Система команд	11
2.1.	Система операций и форматы	11
2.2.	Таблица команд	13
2.3.	Пояснения к некоторым командам таблицы 1	17
2.3.1.	Безадресные команды десятичной коррекции	17
2.3.2.	Умножение	17
2.3.3.	Деление	17
2.3.4.	Цикл	18
2.3.5.	Команды безусловной передачи управления	18
3.	Внешние устройства	19
3.1.	Диспетчер внешних устройств	19
3.2.	Контроллер клавиатуры	20
3.3.	Символьный дисплей	22
3.4.	Таймер-2	23
3.4.1.	Режимы работы Таймера-2	23
3.4.2.	Формат и назначение разрядов регистра TSCR	24
3.4.3.	Особенности организации доступа к 16-разрядным регистрам по 8-разрядному интерфейсу	25
3.5.	Таймер-5	26
3.5.1.	Режимы работы	27
3.5.2.	Форматы регистров состояния/управления	28
3.5.3.	Формирование выходного сигнала	29
3.5.4.	Буферирование старшего байта	29
3.5.5.	Описание режимов работы	30
3.5.6.	Двухканальный цифровой осциллограф	32
3.6.	Контроллер матричной клавиатуры и 7-сегментной динамической индикации	33
3.7.	Цветной графический дисплей	37

II	Лабораторный практикум	40
4.	Лабораторная работа № 1.1. Знакомство с моделью.	
	Программирование простого разветвляющегося процесса	40
4.1.	Порядок выполнения работы	40
4.2.	Содержание отчета	41
4.3.	Контрольные вопросы	41
4.4.	Варианты задания	42
5.	Лабораторная работа № 1.2	
	Программирование цикла	43
5.1.	Пример 2	43
5.2.	Задание 2	44
5.3.	Содержание отчета	45
5.4.	Контрольные вопросы	45
6.	Лабораторная работа № 1.3	
	Подпрограммы и стек	46
6.1.	Пример 3	47
6.2.	Задание 3	47
6.3.	Содержание отчета	47
6.4.	Контрольные вопросы	49
7.	Лабораторная работа № 1.4	
	Программирование внешних устройств	50
7.1.	Задания	50
7.2.	Порядок выполнения работы	52
7.3.	Содержание отчёта	52
7.4.	Контрольные вопросы	52
8.	Лабораторная работа № 2.1.	
	Арифметические операции с многобайтовыми данными	53
8.1.	Кодирование числовых данных	53
8.2.	Выполнение сложения и вычитания с «длинными» данными	53
	8.2.1. Упакованный формат	54
	8.2.2. Распакованный формат	54
8.3.	Задания	55
8.4.	Содержание отчёта	56
9.	Лабораторная работа № 2.2	
	Программирование систем контроля времени	58
9.1.	Возможности таймерных систем микроконтроллеров и их программных моделей в fN8	58
9.2.	Рекомендации по выполнению лабораторной работы	60
9.3.	Содержание отчёта	62
9.4.	Контрольные вопросы	62
9.5.	Варианты заданий	63

Список сокращений

ВПС – верхний предел счёта;

ВУ – внешние устройства;

ОЗУ – оперативное запоминающее устройство;

РВУ – регистр(ы) внешних устройств;

РОН – регистр(ы) общего назначения;

ШИМ – широтно-импульсная модуляция;

ШИМ ФК – широтно-импульсная модуляция с фазовой коррекцией;

ЭВМ – электронная вычислительная машина;

Часть I

Описание модели ЭВМ

1. Архитектура учебной ЭВМ fN8

1.1. Основные характеристики модели

- архитектура фон Неймана (общее ОЗУ для программы и данных);
- формат ячеек – 1 байт;
- объём ОЗУ – 1024 байта;
- стек – в ОЗУ;
- число РОН – 8;
- адресное пространство ввода/вывода – 2^7 адресов;
- число векторов прерываний – 8.

Структурная схема fN8 показана на рис. 1.

1.2. Организация адресного пространства

В модели fN8 реализовано три адресных пространства:

- ОЗУ – 1024 байта;
- РОН – 8 байт;
- РВУ – 128 байт.

При проектировании форматов команд стремились обеспечить для всех команд одинаковую длину – 16 бит. В форматах команд ввода/вывода предусмотрено 7-битовое поле адреса, а в регистровых командах – 3-битовое (см. форматы 10, 11, 12 и 5 на рис. 2). Это позволяет адресовать в командах весь объём пространств ввода/вывода и регистров. Однако в форматах команд, адресующих ячейки ОЗУ (6, 7, 8, 9) поле адреса составляет всего 8 бит. Разрядности счётчика команд PCL и указателя стека SPL так же составляют 8 бит. Следовательно, в этом случае поле адреса таких команд и содержимое регистров PCL и SPL определяет только *смещение в соответствующем сегменте* пространства ОЗУ; очевидно, размер сегмента составляет 256 байт.

В адресном пространстве ОЗУ размещается четыре сегмента с адресами:

Сегмент 0 – 0x000 – 0x0FF

Сегмент 2 – 0x200 – 0x2FF

Сегмент 1 – 0x100 – 0x1FF

Сегмент 3 – 0x300 – 0x3FF

причём среди этих сегментов необходимо выделить сегменты кода, данных и стека.

Для управления размещением сегментов в процессоре предусмотрен специальный программно-доступный регистр SR, в котором SR[1:0] определяет номер сегмента кода (CS), SR[3:2] – номер сегмента данных (DS), SR[5:4] – номер сегмента стека (SS). По умолчанию CS = 00, DS = 10 и SS = 11, то есть под программу отводятся сегменты 0 и 1, под данные – сегмент 2, а под стек – сегмент 3.

Процессор

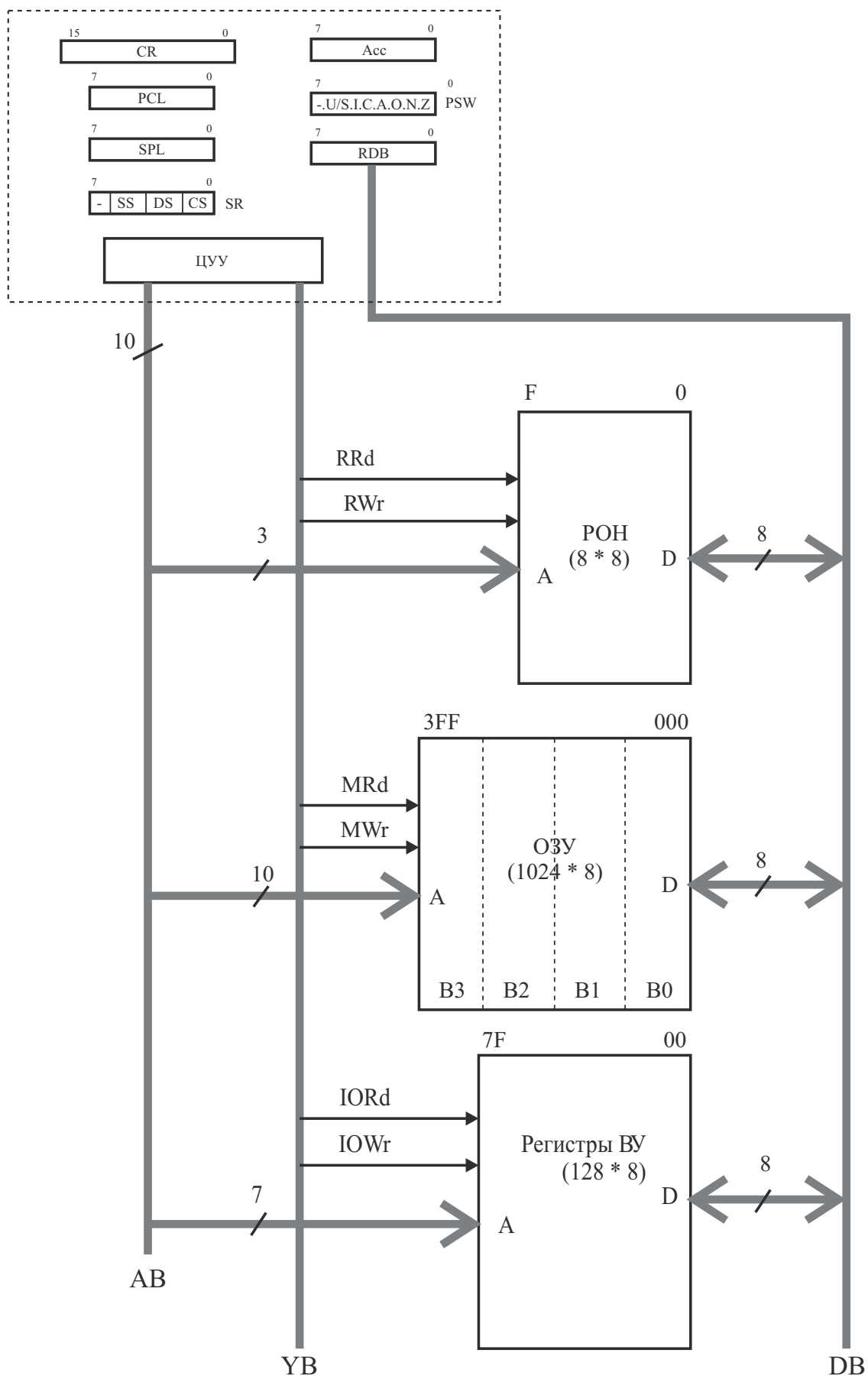


Рисунок 1. Структурная схема fN8

Таблица векторов прерываний размещается в сегменте 0 по адресам 0x00 .. 0x0F и может содержать до восьми векторов, причём *вектор 0 зарезервирован за RESET*. Каждый вектор занимает два байта и содержит 10-разрядный адрес обработчика прерывания. Значение вектора 0 по умолчанию (точка старта программы) – 0x0010.

Адресное пространство ввода/вывода составляет 128 байт, в нём может располагаться до 128 регистров внешних устройств¹ (ВУ). Разделение адресного пространства между подключаемыми ВУ осуществляется путём присвоения каждому устройству базового адреса, кратного 16. Таким образом, одновременно к системе можно подключить до 8 ВУ, каждое из которых может использовать до 16 адресов пространства ввода/вывода.

1.3. Директивы компилятора

При написании программы на языке Ассемблер можно пользоваться следующими директивами компилятора:

- .c <сегмент> – выбирает текущий сегмент компиляции;
- .org <адрес> – изменяет текущий адрес компиляции;
- .db <байт, байт, ... байт> – загрузка констант размером в 1 байт в текущий сегмент начиная с текущего адреса компиляции;
- .dw <слово, метка, ... слово> – загрузка констант размером в 2 байта в текущий сегмент начиная с текущего адреса компиляции.

В процессе компиляции возможна загрузка констант в сегмент данных (например, таблицы ASCII-кодов символов). По умолчанию сегменты «0» и «1» отведены для кода, сегмент «2» – для данных, а сегмент «3» – для стека. Если (лучше в конце текста программы) поставить директиву **.c 2**, то компилятор продолжит компиляцию фактически в сегмент данных. Например, если требуется разместить таблицу 7-сегментных кодов² десятичных цифр в сегменте данных начиная с адреса 0x90, то в конце текста программы можно добавить такой фрагмент:

```
.c 2
.org 0x90
.db 0x3F, 0x06, 0x5B, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77
```

Использование этих директив позволяет изменить установленную по умолчанию точку старта программы – 0x10, например, на адрес 0x20 (a). Ещё проще поставить в нужном месте метку, например, *Start:* и объявить её точкой старта (b).

a)	b)
.org 0	.org 0
.db 0x20, 0x0	.dw Start

Эти же директивы позволяют автоматизировать заполнение **таблицы векторов прерываний**. Таблица векторов прерываний в fN8 размещается в младших адресах памяти 0x000 – 0x00F, каждый вектор занимает два байта. Вектор 0 определяет точку

¹При соответствующей организации схемы ВУ допускается присваивание одинакового адреса двум различным регистрам, из которых один является регистром ввода, а другой – регистром вывода

²В этом примере сегменту **A** индикатора соответствует младший бит кода.

старта (по умолчанию — 0x010), вектора 1, 2, 3 и 4 по умолчанию присваиваются клавиатуре, таймеру-2, таймеру-5 и контроллеру 7-сегментной индикации соответственно. Остальные вектора (5 — 7) можно использовать для других разрабатываемых внешних устройств или для подключения нескольких экземпляров одинаковых ВУ с разными векторами. Адрес младшего байта вектора определяется как его удвоенный номер.

Если поставить метки в начале всех обработчиков прерываний, то загрузка векторов прерываний может выглядеть следующим образом (для случая, когда вектора определяются последовательно):

```
.org 2  
.dw IntKey, IntTim2, IntTim5, Int7Seg
```

По адресам неиспользуемых векторов в таблице векторов прерываний сохраняются значения 0. Если в системе используются те ВУ, вектора прерываний которых по умолчанию следуют не подряд, то можно или при подключении поменять им вектора в желаемом порядке или в директиве заполнения таблицы по неиспользуемым адресам записывать 0, например:

```
.org 2  
.dw IntKey, 0, 0, Int7Seg ,
```

если подключать только контроллер клавиатуры и контроллер 7-сегментной индикации и оставить им назначаемые по умолчанию вектора.

1.4. Ресурсы процессора

Вне адресных пространств – в процессоре – расположены программно-доступные объекты:

- Асс[7:0] – аккумулятор;
- PCL[7:0] – счётчик команд;
- SPL[7:0] – указатель стека;
- SR[7:0] – сегментный регистр;
- PSW[7:0] – регистр слова состояния процессора;

Регистр слова состояния процессора включает следующие флаги:

PSW[0] = Z – нулевой результат;
PSW[1] = N – отрицательный результат;
PSW[2] = O – арифметическое переполнение;
PSW[3] = AC – дополнительный перенос;
PSW[4] = C – перенос;
PSW[5] = I – разрешение прерывания;

Объекты процессора CR[15:0] – регистр команд и DR[7:0] – регистр данных не являются программно-доступными и отображают код выполняемой команды и значение второго операнда бинарной операции АЛУ соответственно.

1.5. Подсистема прерываний fN8

В модели fN8 предусмотрен механизм векторных внешних прерываний. Внешние устройства формируют запросы на прерывания, которые поступают на входы *контроллера прерываний*. При подключении ВУ, способного формировать запрос на прерывание, ему ставится в соответствие номер входа контроллера прерываний — **вектор прерывания**, принимающий значение в диапазоне [1..7].

Контроллер передает вектор, соответствующий запросу, процессору, который начинает процедуру обслуживания прерывания.

Каждому из контролируемых системой прерываний событий должен соответствовать т. н. *обработчик прерывания* — подпрограмма, вызываемая при возникновении события конкретного прерывания.

Механизм прерываний, реализованный в fN8, поддерживает *таблицу векторов прерываний*, которая создается (1.3) в оперативной памяти, может содержать до 8 векторов и располагается по адресам 00 .. 0F нулевого сегмента. Каждый вектор занимает 2 байта памяти и содержит 10-разрядный адрес начала обработчика соответствующего события. Вектор 0 (ячейки 00 и 01) зарезервирован за точкой старта программы по кнопке *Запуск*.

Процессор начинает обработку прерывания³, завершив текущую команду. При этом он

- 1) получает от контроллера вектор прерывания;
- 2) формирует и помещает в верхушку стека двухбайтовое слово, 10 младших разрядов которого — адрес возврата (текущее состояние CS.PCL), а 6 старших — флаги PSW[5:0];
- 3) сбрасывает в «0» флаг разрешения прерывания I;
- 4) извлекает из таблицы векторов прерываний адрес обработчика, соответствующий обслуживаемому вектору, и помещает его в CS.PCL, осуществляя тем самым переход на подпрограмму обработчика прерывания.

Таким образом, вызов обработчика прерывания, в отличие от вызова подпрограммы связан с помещением в стек не только адреса возврата, но и текущего значения вектора флагов. Поэтому последней командой подпрограммы обработчика должна быть команда IRET, которая не только возвращает в PC адрес команды, перед выполнением которой произошло прерывание но и восстанавливает те значения флагов, которые были в момент перехода на обработчик прерывания.

Не всякое событие, которое может вызвать прерывание, приводит к прерыванию текущей программы.

В состав процессора входит программно-доступный флаг I разрешения прерывания. При $I = 0$ процессор не реагирует на запросы прерываний. После сброса процессора флаг I так же сброшен и все прерывания запрещены. Для того, чтобы разрешить прерывания, следует в программе выполнить команду EI (**Enable Interrupt**).

Выше отмечалось, что при переходе на обработчик прерывания флаг I автоматически сбрасывается, в этом случае прервать обслуживание одного прерывания другим прерыванием нельзя. По команде IRET значение флагов восстанавливается, в том числе

³Если прерывания разрешены

вновь устанавливается $I = 1$, следовательно в основной программе прерывания опять разрешены.

Если требуется разрешить прерывания (другие!) в обработчике прерывания, достаточно в нём выполнить команду EI. Контроллер прерываний и процессор на аппаратном уровне блокируют попытки запустить прерывание, если его обработчик начал, но не завершил работу.

Таким образом, флаг I разрешает или запрещает все прерывания системы. Если требуется выборочно разрешить некоторое подмножество прерываний, используются программно-доступные флаги разрешения формирования запросов прерываний непосредственно на внешних устройствах.

Как правило каждое ВУ, которое может вызвать прерывания, содержит в составе своих регистров управления разряды флагов разрешения прерываний, по умолчанию установленных в «0». Если оставить некоторые (или все) эти флаги в «0», то внешнему устройству будет запрещено формировать соответствующие запросы контроллеру прерываний.

Иногда бывает удобно (например, в режиме отладки) иметь возможность вызвать обработчик прерывания непосредственно из программы. Поэтому в системах команд многих ЭВМ, в том числе и fN8, имеются команды вызова прерываний — $INT\ n$, где n — вектор прерывания. Процессор, выполняя команду $INT\ n$, производит те же действия, что и при обработке прерывания с вектором n .

Характерно, что с помощью команды $INT\ n$ можно вызвать обработчик прерывания даже в том случае, когда флаг разрешения прерывания сброшен.

2. Система команд

2.1. Система операций и форматы

Система команд включает в себя следующие операции: арифметические и логические операции над аккумулятором (Асс) и ячейкой памяти или регистром с размещением результата в Асс, команды управления битами, команды пересылки, ввода/вывода, передачи управления (включая вызовы подпрограмм), управление прерываниями.

Адресация в командах с ячейками ОЗУ — прямая и непосредственная. В регистровых командах — прямая, косвенная и несколько вариантов автоиндексной адресации. Адресация в командах ввода/вывода — только прямая. Кроме того, возможна адресация отдельных битов в любой ячейке сегмента данных ОЗУ или в любом РВУ.

Все команды имеют размер 16 бит. Большинство команд при этом являются одноадресными.

Форматы всех команд fN8 показаны на рис. 2

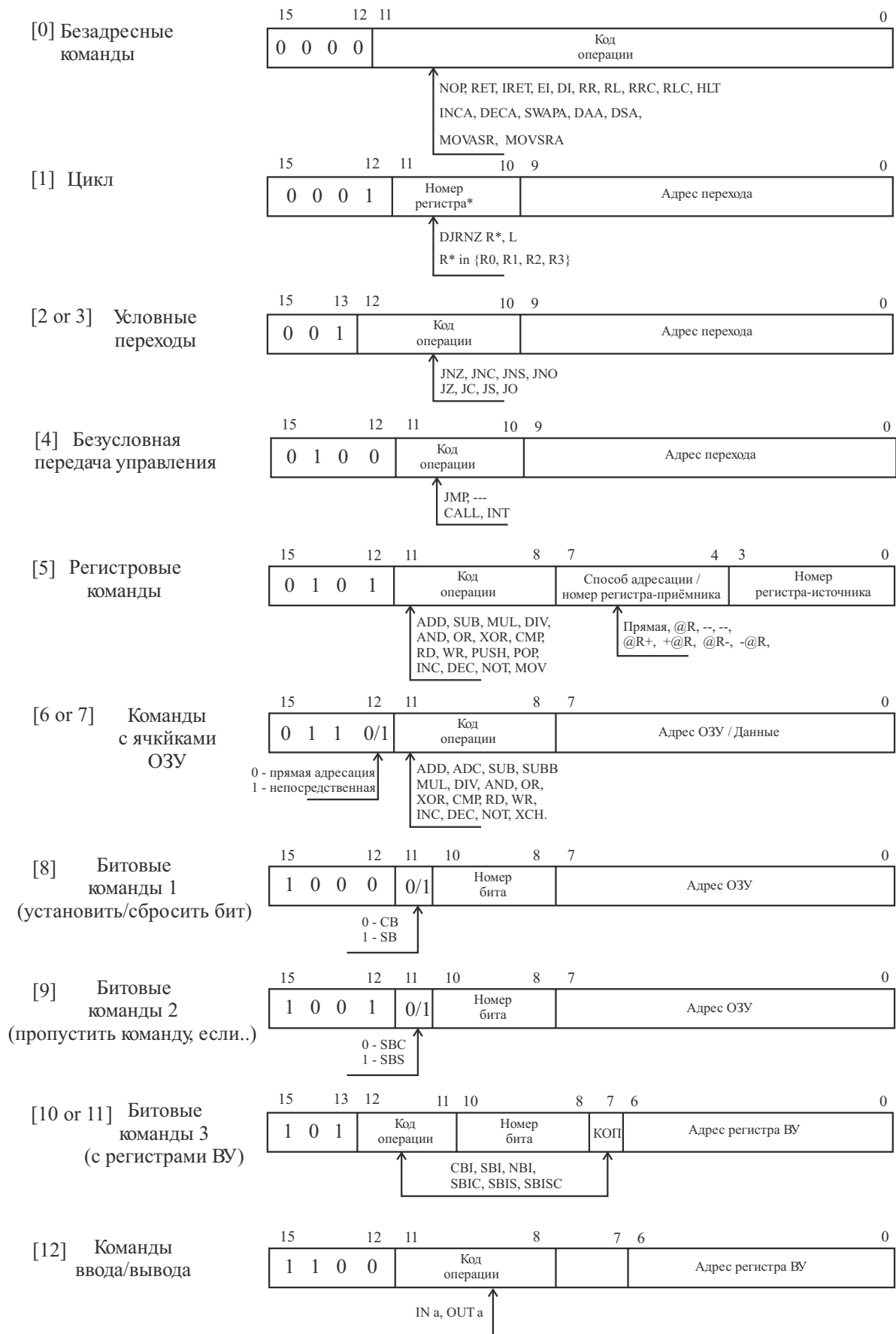


Рисунок 2. Форматы команд fN8

2.2. Таблица команд

В таблице приняты следующие обозначения:

Асс – содержимое аккумулятора;

DD – содержимое ячейки памяти или непосредственный операнд;

R – содержимое регистра общего назначения R;

R* – содержимое регистра или косвенно адресуемой через регистр ячейки памяти;

p – префикс перед именем регистра, определяющий способ адресации;

A – адрес ячейки памяти данных;

M(A) – содержимое ячейки памяти данных по адресу A;

C – флаг PSW[4] переноса(заёма)

SPL – содержимое указателя стека;

L – адрес перехода (метка или абсолютный);

CR – содержимое регистра команд;

RIO – содержимое регистра внешнего устройства;

a – номер (адрес) регистра внешнего устройства;

b – номер бита в байте;

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
Безадресные команды [Код 0]							
NOP	Нет операции	PCL := PCL + 1	-	-	-	-	-
RET	Возврат из подпрограммы	PCL := M(SS.SPL); Inc(SPL); CS := M(SS.SPL)[1:0]; Inc(SPL)	-	-	-	-	-
IRET	Возврат из прерывания	PCL := M(SS.SPL); Inc(SPL); PSW[5:0].CS := M(SS.SPL); Inc(SPL)	-	-	-	-	-
EI	Разрешить прерывание	FI := 1	-	-	-	-	-
DI	Запретить прерывание	FI := 0	-	-	-	-	-
RR	Сдвиг аккумулятора правый циклический	Acc[7:0] := Acc[0].Acc[7:1]; FC := Acc[0]	-	-	-	-	-
RL	Сдвиг аккумулятора левый циклический	Acc[7:0] := Acc[6:0].Acc[7]; FC := Acc[7]	-	-	-	-	-
RRC	Сдвиг аккумулятора правый через перенос	Acc[7:0] := FC.Acc[7:1]; FC := Acc[0]	-	-	-	-	-
RLC	Сдвиг аккумулятора левый через перенос	Acc[7:0] := Acc[6:0].FC; FC := Acc[7]	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
NOTA	Инверсия аккумулятора	$Acc := \overline{Acc}$	0	0	0	+	+
INCA	Инкремент аккумулятора	$Acc := Acc + 1$	-	+	+	+	+
DECA	Декремент аккумулятора	$Acc := Acc - 1$	-	+	+	+	+
SWAPA	Обмен тетрадами аккумулятора	$Acc[7:4] \leftrightarrow Acc[3:0]$	-	-	-	-	-
DAA	Десятичная коррекция сложения	См. раздел 2.3.1	-	+	+	+	+
DSA	Десятичная коррекция вычитания	См. раздел 2.3.1	-	+	+	+	+
MOVSP	Загрузка SPL	$SPL := Acc;$	-	-	-	-	-
MOVAPSW	Прочитать PSW	$Acc := PSW$	-	-	-	-	-
MOVASR	Прочитать SR	$Acc := SR$	-	-	-	-	-
MOVSRA	Загрузить SR	$SR := Acc$	-	-	-	-	-
HLT	Стоп	Прекратить командные циклы	-	-	-	-	-
<i>Цикл [Код 1]</i>							
DJRNZ R^c, L	Цикл	$R^c := R^c - 1;$ if $R^c \neq 0$ then goto L	-	-	-	-	-
$R^c \in \{R0, R1, R2, R3\}$							
<i>Команды условных переходов [Код 2 or 3]</i>							
JNZ L	Переход, если не ноль	if Z=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNC L	Переход, если не перенос	if C=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNN L	Переход, если не отрицательно	if N=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNO L	Переход, если не переполнение	if O=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JZ L	Переход, если ноль	if Z=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JC L	Переход, если перенос	if C=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JN L	Переход, если отрицательно	if N=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JO L	Переход, если переполнение	if O=1 then CS.PCL := CR[9:0]	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
Команды безусловной передачи управления [Код 4]							
JMP L CALL L	Безусловный переход Вызов подпрограммы	CS.PCL := CR[9:0] Dec(SPL); M(SS.SPL)[1:0] := CS; Dec(SPL); M(SS.SPL) := PCL; CS.PCL := CR[9:0]	-	-	-	-	-
INT v	Вызов прерывания	Dec(SPL); M(SS.SPL) := PSW[5:0].CS; Dec(SPL); M(SS.SPL) := PCL; PCL := M(2v); CS := M(2v+1)[1:0]; FI := 0	-	-	-	-	-
v = CR[2:0] – вектор прерывания							
Регистровые команды [Код 5] (Для команд ADC и SUBB – [Код F])							
ADD pR	Сложение	Acc := Acc + R*	+	+	+	+	+
ADC pR	Сложение с переносом	Acc := Acc + R* + C	+	+	+	+	+
SUB pR	Вычитание	Acc := Acc – R*	+	+	+	+	+
SUBB pR	Вычитание с заёмом	Acc := Acc – R* – C	+	+	+	+	+
MUL pR	Умножение	R7.Acc := Acc × R*	-	-	-	-	-
DIV pR	Деление	Acc := Acc : R*	-	-	-	-	+
AND pR	Конъюнкция	Acc := Acc & R*	0	0	0	+	+
OR pR	Дизъюнкция	Acc := Acc ∨ R*	0	0	0	+	+
XOR pR	Неравнозначность	Acc := Acc ⊕ R*	0	0	0	+	+
CMP pR	Сравнение	R* – Acc	+	+	+	+	+
RD pR	Чтение	Acc := R*	-	-	-	-	-
WR pR	Запись	R* := Acc	-	-	-	-	-
XCH pR	Обмен	R* ⇔ Acc	-	-	-	-	-
PUSH R	Поместить в стек	Dec(SPL); M(SPL) := R;	-	-	-	-	-
POP R	Извлечь из стека	R := M(SPL); Inc(SPL)	-	-	-	-	-
INC R	Инкремент	R := R + 1;	-	+	+	+	+
DEC R	Декремент	R := R – 1;	-	+	+	+	+
NOT R	Инверсия	R := \overline{R} ;	0	0	0	+	+
MOV R _r , R _t	Копирование	R _r := R _t	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
Команды с ячейками ОЗУ [Код 6 or 7]							
Код 6 – прямая адресация (A), 7 – непосредственная (#A)							
ADD A	Сложение	Acc := Acc + DD	+	+	+	+	+
ADC A	Сложение с переносом	Acc := Acc + DD + FC	+	+	+	+	+
SUB A	Вычитание	Acc := Acc – DD	+	+	+	+	+
SUBB A	Вычитание с заёмом	Acc := Acc – DD – FC	+	+	+	+	+
MUL A	Умножение	R7.Acc := Acc × DD	-	-	-	-	-
DIV A	Деление	Acc := Acc : DD	-	-	-	-	+
AND A	Конъюнкция	Acc := Acc & DD	0	0	0	+	+
OR A	Дизъюнкция	Acc := Acc ∨ DD	0	0	0	+	+
XOR A	Неравнозначность	Acc := Acc ⊕ DD	0	0	0	+	+
CMP A	Сравнение	DD – Acc	+	+	+	+	+
RD A	Чтение	Acc := DD	-	-	-	-	-
WR A	Запись	M(A) := Acc	-	-	-	-	-
XCH A	Обмен	M(A) ⇔ Acc	-	-	-	-	-
INC A	Инкремент	M(A) := M(A) + 1	-	+	+	+	+
DEC A	Декремент	M(A) := M(A) – 1	-	+	+	+	+
NOT A	Инверсия	M(A) := \overline{DD}	0	0	0	+	+
Битовые команды 1 [Код 8]							
CB A,b	Сбросить бит	M(A)[b] := 0	-	-	-	-	-
SB A,b	Установить бит	M(A)[b] := 1	-	-	-	-	-
Битовые команды 2 [Код 9]							
SBC A,b	Пропустить следующую команду, если бит сброшен	if M(A)[b]=0 then PCL := PCL + 2	-	-	-	-	-
SBS A,b	Пропустить следующую команду, если бит установлен	if M(A)[b]=1 then PCL := PCL + 2	-	-	-	-	-
Битовые команды 3 [Код A or B]							
CBI a,b	Сбросить бит	RIO(a)[b] := 0	-	-	-	-	-
SBI a,b	Установить бит	RIO(a)[b] := 1	-	-	-	-	-
NBI a,b	Инвертировать бит	$RIO(a)[b] := \overline{RIO(a)[b]}$	-	-	-	-	-
SBIC a,b	Пропустить следующую команду, если бит сброшен	if RIO(a)[b]=0 then PCL := PCL + 2	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
SBIS a,b	Пропустить следующую команду, если бит установлен	if RIO(a)[b]=1 then PCL := PCL + 2	-	-	-	-	-
SBISC a,b	Пропустить следующую команду, если бит установлен и сбросит бит	if RIO(a)[b]=1 then (PCL := PCL + 2; RIO(a)[b] := 0)	-	-	-	-	-
<i>Команды ввода/вывода [Код C]</i>							
IN a	Ввод	Acc := RIO(a)	-	-	-	-	-
OUT a	Вывод	RIO(a) := Acc	-	-	-	-	-

2.3. Пояснения к некоторым командам таблицы 1

2.3.1. Безадресные команды десятичной коррекции

DAA – десятичная коррекция байта в Асс после сложении (кодировка «8421»)

$$\text{if } (B_L > 9) \text{ or } (AC) \text{ then } B := B + 0x06; \quad (1)$$

$$\text{if } (B_H > 9) \text{ or } (C) \text{ then } B := B + 0x60; \quad (2)$$

DSA – десятичная коррекция байта в Асс после вычитании (кодировка «8421»)

$$\text{if } (AC) \text{ then } B := B - 0x06; \quad (3)$$

$$\text{if } (C) \text{ then } B := B - 0x60; \quad (4)$$

Здесь B – предварительный результат операции сложения (вычитания), B_H – старшая тетрада B , B_L – младшая тетрада B , AC – перенос (заём) из младшей тетрады, C – перенос (заём) из байта.

2.3.2. Умножение

Сомножители рассматриваются как целые беззнаковые числа. Содержимое аккумулятора можно умножить на содержимое регистра, прямо или косвенно адресуемой ячейки памяти или 8-разрядную константу. Формат произведения – два байта, поэтому младший байт размещается в аккумуляторе Асс, а старший – в регистре R7.

2.3.3. Деление

Целочисленное беззнаковое деление: байт Асс делится на байт делителя, целая часть частного помещается в Асс. В качестве делителя можно использовать непосредственный операнд, содержимое регистра, содержимое прямо или косвенно адресуемой ячейки памяти. Если содержимое Асс меньше делителя, то результат деления равен 0.

2.3.4. Цикл

DJRNZ R,L – декремент регистра и проверка. Если после декремента содержимое регистра $\neq 0$, то осуществляется переход по указанному адресу, иначе – на следующую команду. Можно использовать для организации циклов. Работает только с регистрами R0, R1, R2, R3.

2.3.5. Команды безусловной передачи управления

CALL L – вызов подпрограммы по прямому адресу. Использует две ячейки стека для размещения 10-разрядного адреса возврата (в старшем байте занято только два младших разряда).

INT v – вызов обработчика прерывания по вектору v . Как и команда **CALL**, сохраняет адрес возврата, а в свободные 6 разрядов старшего байта помещает флаги I, O, C, AC, N, Z. Таблица векторов прерываний в ОЗУ начинается с адреса 0x000, вектор прерываний $v \in \{0, 1, \dots, 7\}$. Адрес обработчика прерываний загружается из ячеек с адресами $(2v)$ и $(2v+1)[1:0]$ в PCL и SR[1:0] соответственно.

3. Внешние устройства

Модели внешних устройств (ВУ), используемые в fN8, реализованы по единому принципу. С точки зрения процессора они представляют собой ряд программно-доступных регистров, лежащих в адресном пространстве ввода/вывода. Размер регистров ВУ совпадает с размером ячеек памяти и регистров данных процессора — один байт.

Доступ к регистрам ВУ осуществляется по командам *IN a*, *OUT a*, где *a* — семиразрядный двоичный адрес регистра ВУ. Таким образом, общий объем адресного пространства ввода/вывода составляет $2^7 = 128$ адресов. Следует помнить, что адресные пространства памяти и ввода/вывода в этой модели разделены.

Разные ВУ содержат различное число программно-доступных регистров, каждому из которых соответствует свой адрес, причем нумерация адресов в каждом ВУ начинается с 0. При подключении ВУ ему средствами *Диспетчера внешних устройств* ставится в соответствие базовый адрес в пространстве ввода/вывода и все адреса регистров ВУ становятся *смещениями* относительно его базового адреса.

3.1. Диспетчер внешних устройств

Окно диспетчера (рис. 3) отображает список ВУ, доступных для подключения в систему. Каждому ВУ предлагается базовый адрес по умолчанию, а тем устройствам, которые работают с прерываниями — и вектор прерывания по умолчанию.

Пользователь может произвольно изменить предлагаемые по умолчанию параметры. При этом надо учитывать, что базовый адрес ВУ в пространстве ввода/вывода всегда должен быть кратным 16, поэтому в окне диспетчера можно задать только старшую цифру шестнадцатеричного базового адреса в диапазоне от 0 до 7. При выборе вектора прерывания следует учитывать, что вектор 0 зарезервирован за сигналом RESET, а остальные вектора от 1 до 7 могут назначаться ВУ произвольно.

Не следует назначать разным устройствам одинаковые базовые адреса и вектора прерывания!

Подключение ВУ осуществляется нажатием кнопки с соответствующим именем в окне диспетчера. Отключение подключённых ВУ в процессе работы fN8 не предусмотрено.

Возможна работа с несколькими одинаковыми ВУ, если при последовательном подключении им будут назначены разные базовые адреса и вектора прерывания.

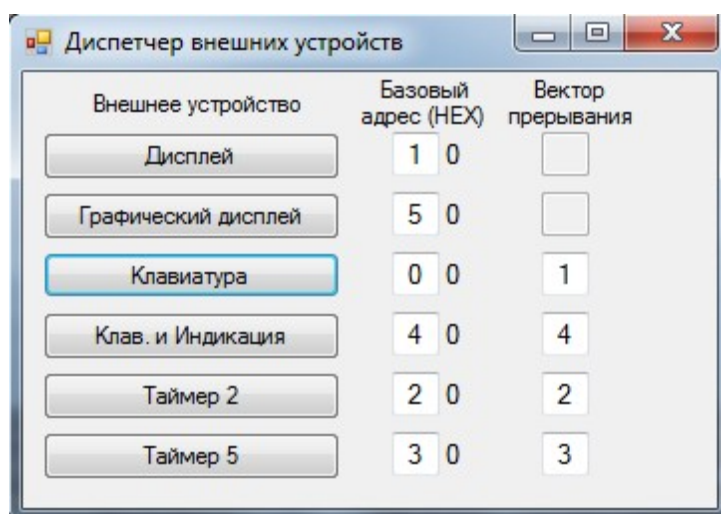


Рисунок 3. Окно диспетчера ВУ

3.2. Контроллер клавиатуры

Контроллер клавиатуры является двоичным функциональным аналогом контроллера клавиатуры, входящего в состав модели десятичной ЭВМ *CompModel* [1, 2]. Контроллер содержит 64-символьный буфер ASCII-кодов нажатых клавиш, три программно-доступных регистра, две кнопки и схему управления (рис. 4).

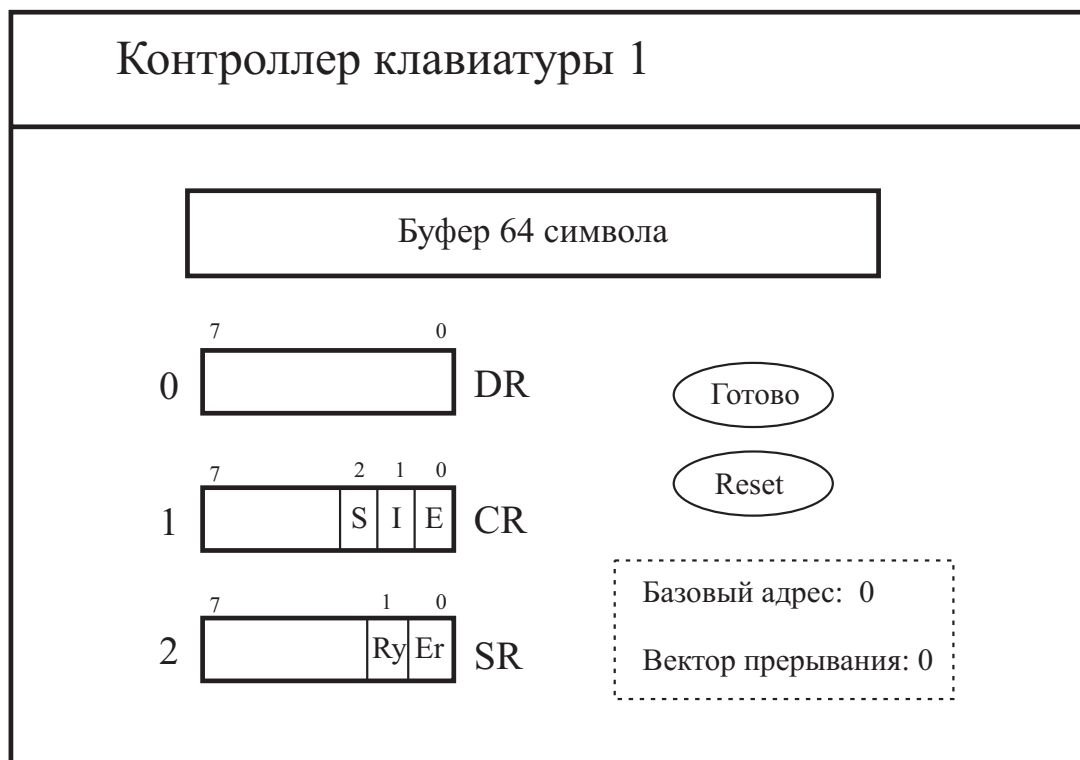


Рисунок 4. Контроллер клавиатуры

В состав контроллера клавиатуры входят три программно-доступных регистра:

- DR (адрес 0) — регистр данных;
- CR (адрес 1) — регистр управления, определяет режимы работы контроллера и содержит следующие флаги:

CR[0] = *E* — флаг разрешения приема кодов в буфер;

CR[1] = *I* — флаг разрешения формирования запроса на прерывание;

CR[2] = *S* — флаг режима ввода.

- SR (адрес 2) — регистр состояния, доступен по чтению и записи, содержит два флага:

SR[0] = Err — флаг ошибки;

SR[1] = Rdy — флаг готовности.

Регистр данных DR доступен только для чтения, через него считываются ASCII-коды из буфера, причем порядок чтения кодов из буфера соответствует порядку их записи в буфер — каждое чтение по адресу 0 автоматически перемещает указатель

чтения буфера. В каждый момент времени DR содержит код символа по адресу указателя чтения буфера.

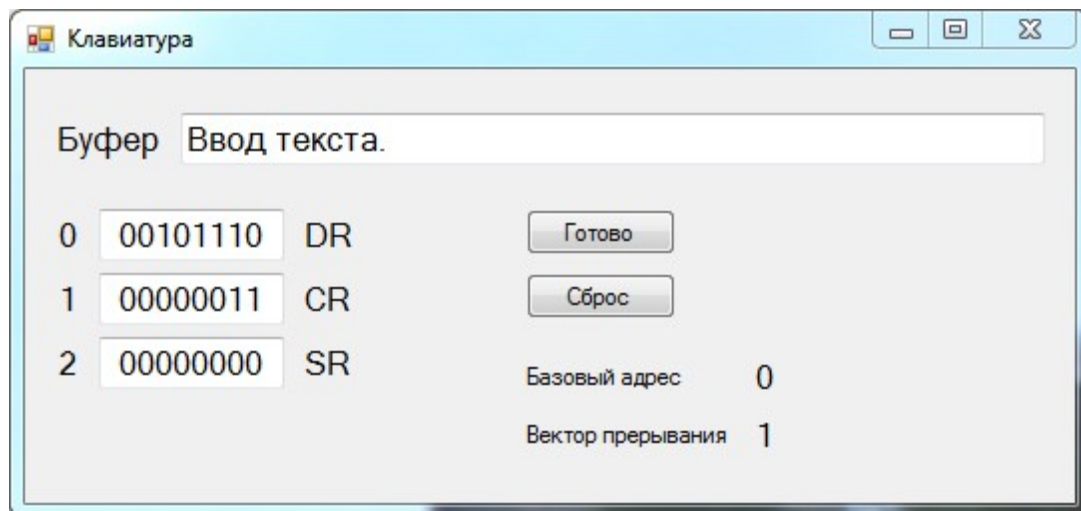


Рисунок 5. Окно обозревателя «Контроллер клавиатуры»

Флаги **регистра управления CR** устанавливаются и сбрасываются программно.

Флаг *E*, будучи установленным, разрешает приём кодов в буфер. При $E = 0$ контроллер игнорирует нажатие на клавиатуре, приём кодов в буфер прекращается. На считывание кодов из буфера флаг *E* влияния не оказывает.

Флаг *I*, будучи установленным, разрешает при определенных условиях формирование контроллером запроса на прерывание. При $I = 0$ запрос на прерывание не формируется.

Состояние флага *S* определяет условие формирования флага готовности Rdy (Ready) в регистре SR. В режиме *посимвольного ввода* ($S = 0$, по умолчанию) флаг Rdy устанавливается после ввода в буфер кода каждого символа, а сбрасывается аппаратно после считывания кода символа из регистра данных DR.

В режиме *строчного ввода* ($S = 1$) флаг Rdy устанавливается только после нажатия в окне обозревателя клавиатуры кнопки *Готово*; сбрасывается флаг Rdy так же, как в режиме посимвольного ввода.

Запрос на прерывание всегда формируется по условию $(I = 1) \& (Rdy = 1)$.

Флаг ошибки Err (Error) в регистре SR устанавливается при попытке ввода в буфер 65-го символа. Ввод 65-го и всех последующих символов блокируется.

Сброс флага Rdy осуществляется автоматически при чтении из регистра DR, флаг Err сбрасывается программно. Кроме того, оба эти флага сбрасываются при нажатии кнопки *Сброс* на панели обозревателя; одновременно со сбросом флагов производится очистка буфера – весь буфер заполняется кодами 00h и указатели записи и чтения устанавливаются в начало буфера.

3.3. Символьный дисплей

Дисплей может отображать символы, задаваемые ASCII-кодами, поступающими на его регистр данных. Окно обозревателя дисплея показано на рис. 6.

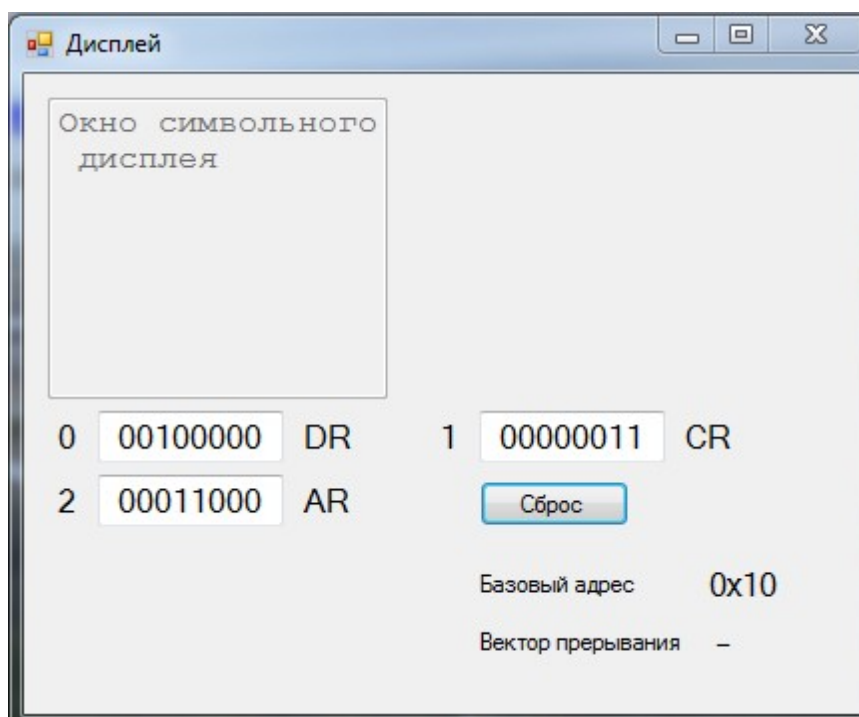


Рисунок 6. Окно обозревателя «Дисплей»

Дисплей включает:

- символьный экран размером 8 строк по 16 символов в строке;
- видеопамять объемом 128 байт (ОЗУ дисплея);
- три программно-доступных регистра:

DR (адрес 0) — регистр данных;

CR (адрес 1) — регистр управления;

AR (адрес 2) — регистр адреса;

Через регистры **адреса AR** и **данных DR**, доступные по записи и чтению, осуществляется доступ к ячейкам видеопамати. При обращении к регистру DR по записи содержимое аккумулятора записывается в DR и в ячейку видеопамати, адрес которой установлен в регистре AR. При чтении DR в аккумулятор загружается содержимое ячейки видеопамати по адресу AR.

Регистр **управления CR** доступен только по записи и содержит два флага:

$CR[0] = E$ — флаг разрешения работы дисплея; при $E = 0$ запись в регистры AR и DR блокируется.

$CR[1] = A$ — флаг автоинкремента адреса; при $A = 1$ содержимое AR автоматически увеличивается на 1 после любого обращения к регистру DR — по записи или чтению. $CR[7 : 2]$ — не используются

3.4. Таймер-2

В состав набора ВУ, которые могут подключаться к fN8 входит несколько таймерных устройств с различными функциональными возможностями. Наиболее простым из них можно считать Таймер-2, функциональная схема которого показана на рис. 7.

Таймер-2 включает:

- 16-разрядный суммирующий счётчик TCNT с программируемым предделителем,
- 16-разрядный регистр TIOR захвата/автозагрузки,
- схему сравнения $TCNT == TIOR$,
- 16-разрядный регистр TSCR управления/состояния,
- буферный регистр старшего байта Buf_H.

3.4.1. Режимы работы Таймера-2

Таймер-2 может работать в одном из четырёх режимов:

Режим 0 – **простой счёт**. В этом режиме состояние TCNT инкрементируется с частотой, определяемой коэффициентом деления предделителя. При переходе счётчика из состояния 0xFFFF в состояние 0x0000 устанавливается флаг переполнения OVF, формируется запрос на прерывание, если установлен флаг разрешения прерывания по переполнению OVIE, счёт продолжается.

Режим 1 – **режим захвата**. Здесь по внешнему (по отношению к Таймеру-2) сигналу текущее состояние TCNT копируется в TIOR, устанавливается флаг захвата ICF и формируется запрос на прерывание при условии ICIE = 1, счёт продолжается. Источником сигнала «Захват» может служить одноимённая кнопка в окне обозревателя Таймера-2 (рис. 8) или установленный программно бит IC в регистре TSCR.

Режим 2 – **сравнение**. В этом режиме предварительно устанавливается произвольное значение в регистр TIOR и с ним сравнивается текущее значение TCNT. При $TCNT = TIOR$ устанавливается флаг сравнения OCF и формируется запрос на прерывание при условии OCIE = 1. Если разряд RT регистра TSCR сброшен, то счёт продолжается далее, а если $RT = 1$, то TCNT обнуляется и счёт начинается с начала. Фактически такой режим аналогичен простому счёту, верхний предел которого определяется содержимым TIOR.

Режим 3 – **автозагрузка**. Этот режим позволяет начинать счёт не с 0, а со значения, установленного в TIOR. Счёт идёт до значения 0xFFFF, устанавливается флаг переполнения OVF, может быть сформирован запрос на прерывание, а в TCNT копируется значение из TIOR.

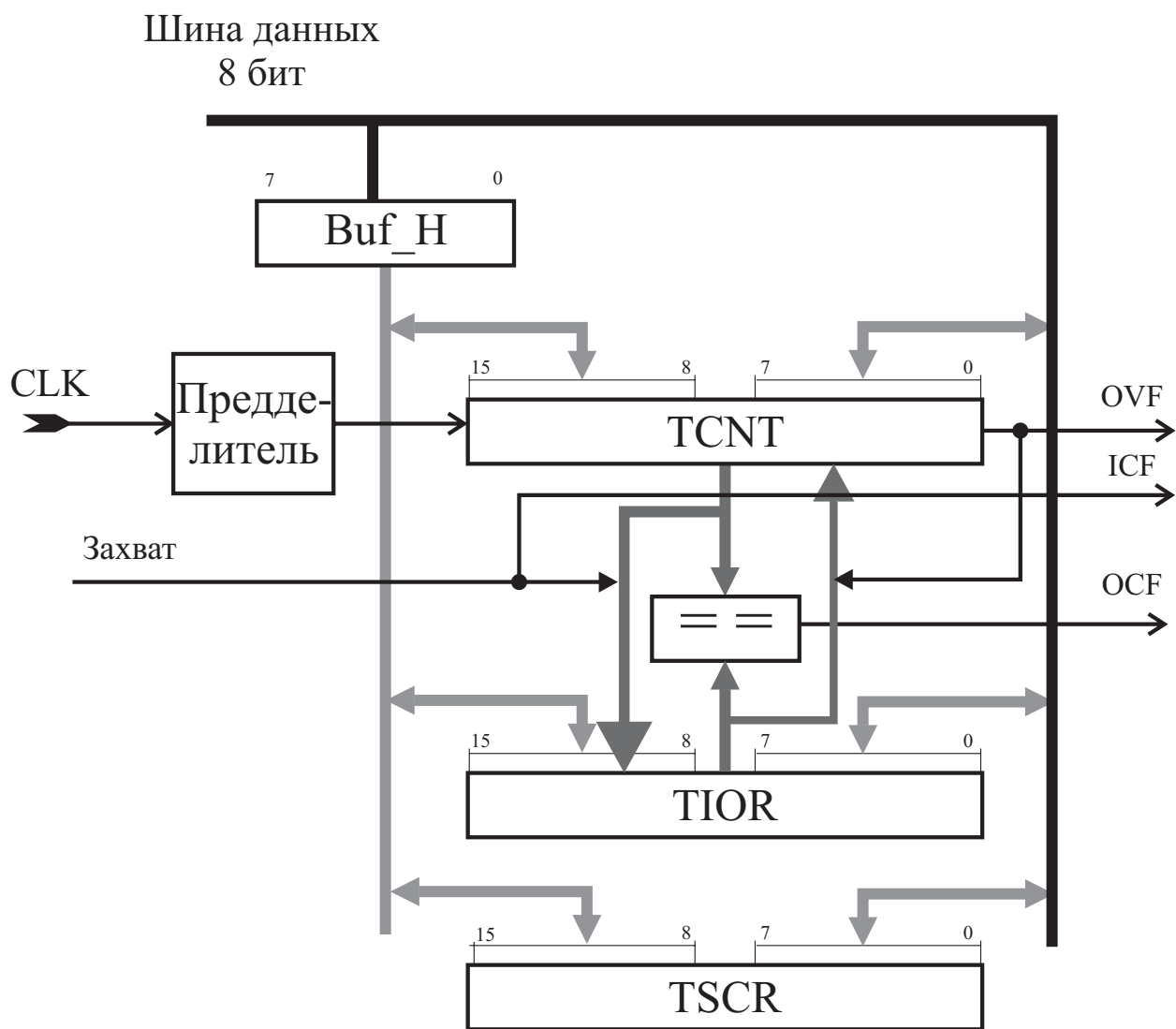


Рисунок 7. Функциональная схема Таймера-2

3.4.2. Формат и назначение разрядов регистра TSCR

№ бита	Имя	Назначение
0	T	Включить таймер
1..2	Code	Режим
3	RT	Сброс TCNT при сравнении
4	OVIE	Разрешить прерывание при переполнении
5	OCIE	Разрешить прерывание при сравнении
6	ICIE	Разрешить прерывание при захвате
7	0	Разрешение загрузки TSCRL
8(0)	IC	Программный «Захват»
9(1)	OVF	Флаг переполнения
10(2)	OCF	Флаг сравнения
11(3)	ICF	Флаг захвата
14(6)..12(4)	k	Коэффициент деления предделителя
15(7)	0	Разрешение загрузки TSCRH

Флаги OVF, OCF, и ICF устанавливаются аппаратно при возникновении соответствующего события и сбрасываются только программно.

Разряд IC – «Захват» может устанавливаться программно или по нажатию кнопки *Захват* в окне обозревателя *Таймера 2*. Установка бита IC в режиме «01» немедленно вызывает захват содержимого TCNT в TIOR, установку флага ICF и при ICIE = 1 – формирование запроса на прерывание. Кроме того, после установки ICF бит IC автоматически сбрасывается. В других режимах работы таймера установка IC игнорируется.

3.4.3. Особенности организации доступа к 16-разрядным регистрам по 8-разрядному интерфейсу

Очевидно, что в адресном пространстве 8-разрядной ЭВМ 16-разрядный регистр должен занимать два соседних адреса, причём обычно младший байт получает чётный адрес, а старший – нечётный больший. Выполнив пару команд обращения, можно прочитать или загрузить 16-разрядный код. Проблемы могут возникнуть, если состояние 16-разрядного регистра может изменяться с достаточно высокой частотой. Так, при считывании, например, значения TCNT = 0x03FF переходящее в 0x0400, один байт может быть прочитан из «старого» кода, а другой – из «нового» (0x0300 или 0x04FF) – ошибка значительно превысит величину младшего разряда!

Чтобы исключить подобные ситуации в таймерных системах принято осуществлять буферирование старшего байта 16-разрядных слов. Чтение⁴ двухбайтового слова следует всегда начинать с младшего байта, при этом старший байт слова одновременно копируется в буферный регистр. При чтении старшего байта слова, его значение извлекается не из регистра, а из буфера. Таким образом, оба считанных байта слова относятся к одному моменту дискретного времени.

Аналогично при записи (см. сноску 4) в двухбайтовое слово сначала следует записывать по старшему адресу, при этом байт записывается в буфер, а старший байт 16-разрядного регистра не меняется. По команде записи в младший байт слова одновременно производится запись в старший байт из буфера, таким образом оба байта слова попадают в 16-разрядный регистр одновременно.

Единственный буфер Buf_H в Таймере-2 является общим для всех трёх 16-разрядных регистров, поэтому, если требуется записать в регистры константы с одинаковыми старшими байтами, достаточно один раз записать константу в буфер.

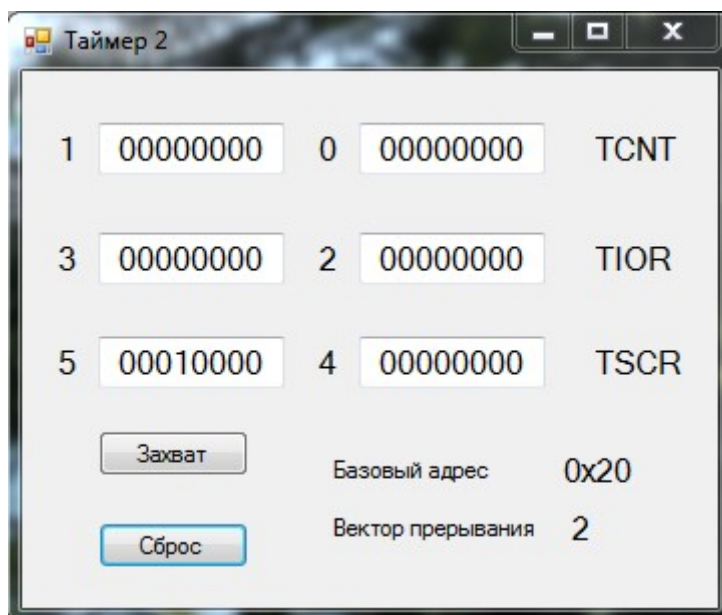


Рисунок 8. Окно обозревателя Таймер-2

⁴ Не забываем, что чтение из регистров ВУ осуществляется командами *IN a*, а запись – *OUT a*.

Следует помнить, что в операции записи байта в TSCRH значение разрядов $b_3 \dots b_1$ безразлично. Эти разряды соответствуют флагам OVF, OCF, и ICF, которые устанавливаются только аппаратно, а при записи любого байта в TSCRH сохраняют свои прежние значения. Сбросить эти флаги можно с помощью команд записи бита.

В регистре управления/состояния TSCR часто требуется изменить отдельные биты, поэтому в механизме загрузки TSCR предусмотрены дополнительные возможности.

Если необходимо произвести запись байтов в TSCR (старшего, потом младшего) следует соблюдать дополнительное условие: старшие биты b_7 этих байтов должны быть «0». Если по адресу TSCRH (0x25) или TSCRL (0x24) производится запись байта с $b_7 = 1$, то это интерпретируется Таймером-2 как команда записи бита в адресуемый регистр, причём разряды $b_3 \dots b_1$ определяют номер бита, а b_0 – устанавливаемое значение⁵.

3.5. Таймер-5

Функциональная схема модуля Таймера-5 показана на рисунке 9, а окно обозревателя – на рисунке 10.

Модуль включает:

- 16-разрядный реверсивный счётчик TCNT (относительные адреса 1, 0),
- предделитель частоты счёта,
- 16-разрядный регистр сравнения OCR (3, 2),
- схему сравнения $CNT == OCR$,
- 16-разрядный регистр захвата ICR (5, 4),
- 8-разрядный буферный регистр старших байтов BUFH,
- два 8-разрядных регистра состояния/управления TSCRL (6) и TSCRH (7),
- блок управления таймером,
- формирователь выходного сигнала.

При подключении к fN8 Таймеру-5 по умолчанию присваивается базовый адрес 0x30⁶, адресное пространство таймера составляет 8 байт (адреса 0x37..0x30).

Таймер-5 может формировать запросы на прерывания по трём различным событиям:

- превышение заданного верхнего предела счёта,
- захват состояния TCNT в ICR
- совпадение значений TCNT и OCR.

Все три события генерируют запрос на прерывание с вектором 3 (по умолчанию, можно изменить), а идентификация конкретного события должна выполняться программно в обработчике прерывания путём анализа соответствующих флагов регистра TSCRH.

⁵В fN8 можно воспользоваться битовыми командами *CBI a,b* или *SBI a,b*.

⁶Напомним, что при необходимости базовый адрес при подключении может быть изменён (см. раздел 3.1).

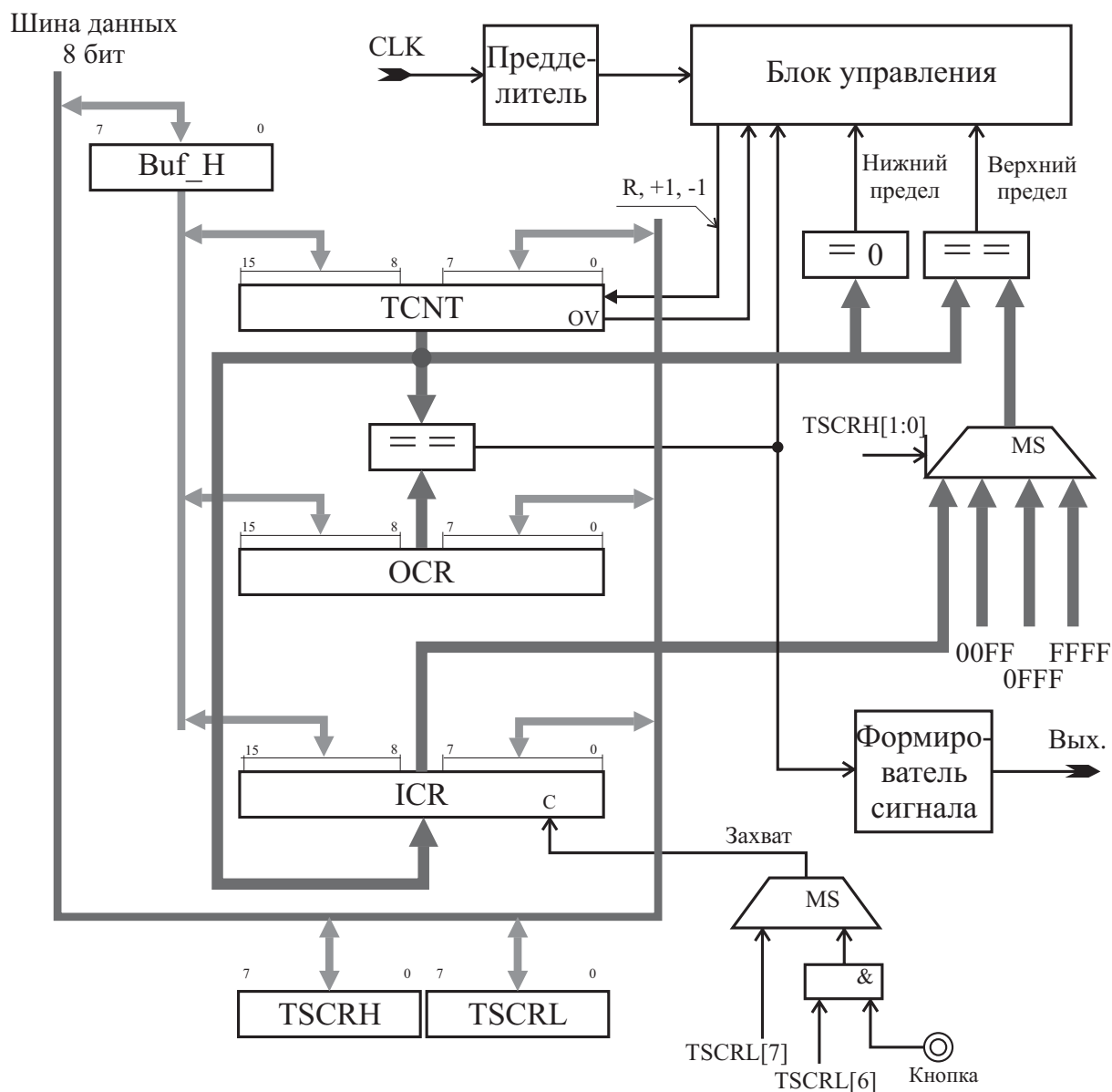


Рисунок 9. Функциональная схема Таймера-5

Верхний предел счёта устанавливается программно (поле TSCRH[1:0]) и может принимать значение одной из трёх констант: 0x00FF, 0x0FFF, 0xFFFF. Кроме того, в качестве значения верхнего предела может выступать содержимое регистра ICR, в этом случае в качестве верхнего предела может быть выбрано любое значение от 0 до 0xFFFF.

3.5.1. Режимы работы

Модуль поддерживает следующие режимы работы:

Режим **0** – **Простой счёт**: инкремент TCNT, формирование флага OVF при переходе *Верхний предел* \rightarrow 0.

Режим **1** – **Сброс TCNT при совпадении** TCNT = OCR, формирование флага ICF.

Режим **2** – **Быстрый ШИМ**;

Режим **3** – **ШИМ с фазовой коррекцией** (ШИМ ФК).

3.5.2. Форматы регистров состояния/управления

	7	6	5	4	3	2	1	0
TSCRL (0x36)	ПЗ	РВЗ	Внешний сигнал	Частота	Режим			

Назначение полей регистра TSCRL:

TSCRL[1:0] – определяет режим работы (0..3);

TSCRL[3:2] – задаёт частоту счёта (управление предделителем):

00 – счётчик выключен,

01 – OSC (частота задающего генератора),

10 – OSC/16,

11 – OSC/64;

TSCRL[5:4] – управление формированием внешнего сигнала (зависит от выбранного режима, см. табл. в разделе [3.5.3](#));

TSCRL[6] – разрешение внешнего захвата;

TSCRL[7] – программный захват.

	7	6	5	4	3	2	1	0
TSCRH (0x37)	ICF	OCF	OVF	ICIE	OCIE	OVIE	Верхний предел	

Назначение полей регистра TSCRH:

TSCRH[1:0] – определяет значение верхнего предела:

00 – ICR

01 – 00FF

10 – 0FFF

11 – FFFF

TSCRH[4:2] – биты *разрешения прерываний* по событиям: внешний захват (ICIE), совпадение TCNT = OCR (OCIE), таймер достиг верхнего предела (OVIE);

TSCRH[7:5] – флаги соответствующих событий: ICF, OCF, OVF.

3.5.3. Формирование выходного сигнала

TSCRL[5:4]	Без ШИМ	Быстрый ШИМ	ШИМ ФК
00	Выход отключён		
01	Инверсия при совпадении TCNT = OCR		
10	Сброс («0») с при совпадении	Сброс при совпадении, установка на верхнем пределе счёта	Сброс при совпадении во время прямого счёта, установка при совпадении во время обратного счёта
11	Установка («1») при совпадении	Установка при совпадении, сброс на верхнем пределе счёта	Установка при совпадении во время прямого счёта, сброс при совпадении во время обратного счёта

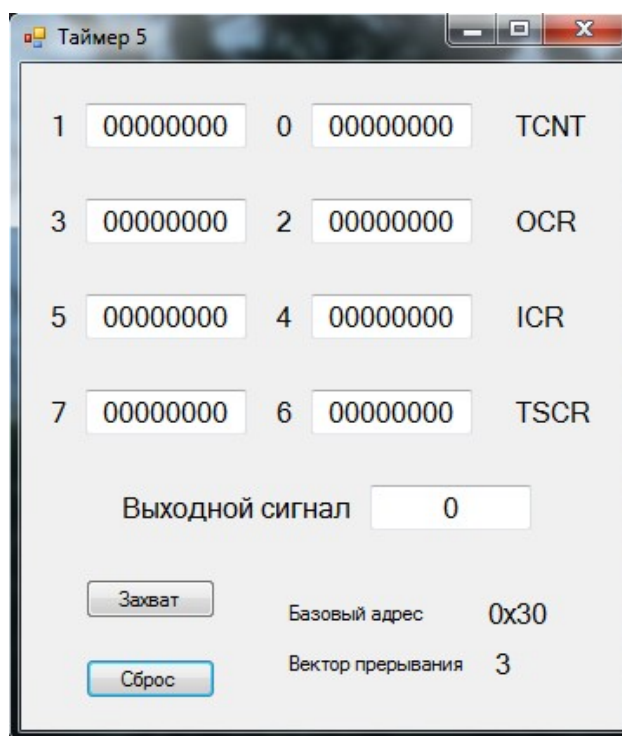


Рисунок 10. Окно обозревателя Таймера-5

3.5.4. Буферирование старшего байта

Для того, чтобы извлекаемые/записываемые данные 16-разрядных регистров, передаваемые по 8-разрядной шине, *относилось к одному моменту времени* используется буферирование старшего байта регистров.

При чтении из регистра программа должна обращаться сначала к его младшему байту. При этом содержимое старшего байта в том же такте автоматически копируется в буфер Buf_H. Команда чтения из старшего байта регистра считывает информацию из Buf_H. Таким образом, оба считанных из регистра байта относятся к одному моменту времени.

При записи следует сначала записывать старший байт, при этом он всегда попадает в Buf_H. По команде записи в регистр младшего байта старший байт одновременно загружается в регистр из Buf_H.

Заметим, что обращение в регистр состояния/управления TSCR не буферизуется – обращение выполняется непосредственно в TSCRH, причём обращаться к TSCRH и TSCRL можно в любом порядке.

3.5.5. Описание режимов работы

Режим 0 – Простой счёт. TCNT икрементируется от нижнего предела (всегда 0) до определённого в поле TSCRH[1:0] верхнего предела счёта⁷ с частотой, определяемой полем TSCRL[3:2]. В момент достижения верхнего предела TCNT сбрасывается в 0x0; устанавливается флаг переполнения OVF; формируется запрос на прерывание с вектором 3, если OVIE = 1; продолжается счёт.

В Режиме 0 можно использовать **функцию захвата** состояния TCNT в регистр ICR. Захват выполняется при нажатии кнопки *Захват*, если установлено разрешение внешнего захвата (TSCRL[6] = 1) или безусловно – по установке флага *Программный захват* (TSCRL[7]). При внешнем захвате устанавливается флаг ICF и может быть сформирован запрос на прерывание с вектором 3, если ICIE = 1.

Примечание. Отдельно следует сказать о захвате в ситуации, когда верхний предел задаётся содержимым регистра ICR. Внешний захват (если TSCRL[6] = 1) выполняется успешно, при этом заданное в ICR значение верхнего предела сохраняется во внутреннем регистре и продолжает определять верхний предел счёта. Программный захват выполняется, при этом значение верхнего предела становится неопределённым.

Режим 1 – Сброс по совпадению. Так же, как и в Режиме 0, выбирается частота счёта, верхний предел и, кроме того, загружается константа в регистр OCR. При совпадении TCNT = OCR устанавливается флаг совпадения OCF; при установленном флаге OCIE формируется запрос на прерывание; TCNT сбрасывается в 0x0 и продолжает счёт; возможно изменение выходного сигнала в соответствии с установками поля TSCRL[5:4] (см. таблицу в разделе 3.5.3).

Очевидно, значение OCR должно быть меньше верхнего предела счёта, иначе совпадение TCNT = OCR не наступит никогда.

В режиме 1 функция захвата работает как по внешнему, так и по программному сигналу.

Режим 2 – Быстрая ШИМ. В этом режиме период импульса ШИМ опреде-

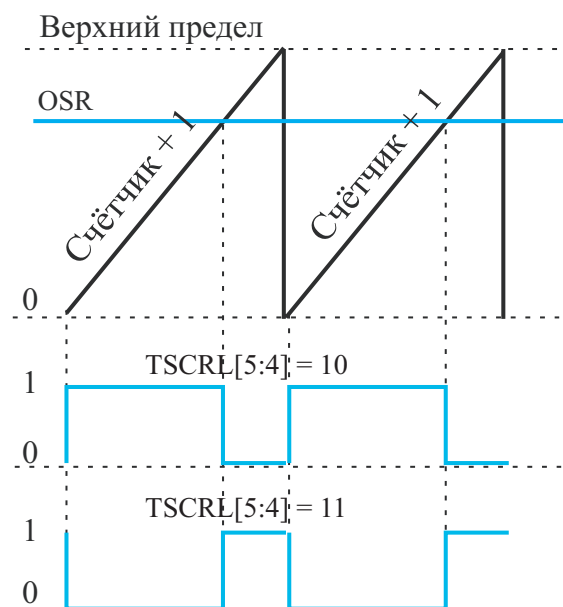


Рисунок 11. Быстрая ШИМ

⁷При TSCRH[1:0] = 00 значение верхнего предела программно задаётся в регистр ICR.

ляется значением верхнего предела (и, разумеется, выбранной тактовой частотой), а длительность импульса – значением регистра OCR. По достижении верхнего предела TCNT сбрасывается в 0 и начинается новый период ШИМ (рис. 11).

Значение регистра OCR должно выбираться всегда *меньше* установленного верхнего предела счёта (ВПС). Тогда в каждом периоде ШИМ можно выделить два события: $TCNT = ВПС$ и $TCNT = OCR$. Поле (TSCRL[5:4]) определяет, по какому из этих двух событий выходной сигнал устанавливается в «1», а по какому – сбрасывается в «0».

В режиме 2 не формируется флаг совпадения OCF, но при достижении верхнего предела счёта устанавливается OVF, что позволяет при необходимости «поймать» момент завершения очередного периода ШИМ.

На фоне генерации ШИМ-импульсов можно захватывать текущее состояние TCNT в регистр ICR программно (команда `sbi 0x36,7`) или по внешнему сигналу (кнопка *Захват* обозревателя), причём внешний захват возможен при условии $TSCRL[6] = 1$. Относительно захвата в случае, когда для задания верхнего предела счёта используется регистр ICR см. Примечание на стр. 30.

Режим 3 – ШИМ с фазовой коррекцией. Для некоторых применений ШИМ важным является расположение импульса в периоде ШИМ. Режим 3 обеспечивает выдачу импульса, симметричного относительно середины периода периода ШИМ при любой длительности (рис. 12).

Период ШИМ ФК определяется временем перехода TCNT ($0 \rightarrow ВПС \rightarrow 0$), то есть при прочих равных условиях он вдвое длиннее периода быстрого ШИМ. Изменение значения выходного сигнала происходит при совпадении $TCNT = OCR$ на участках прямого и обратного счёта TCNT в соответствии с правилами, приведёнными в табл. 3.5.3.

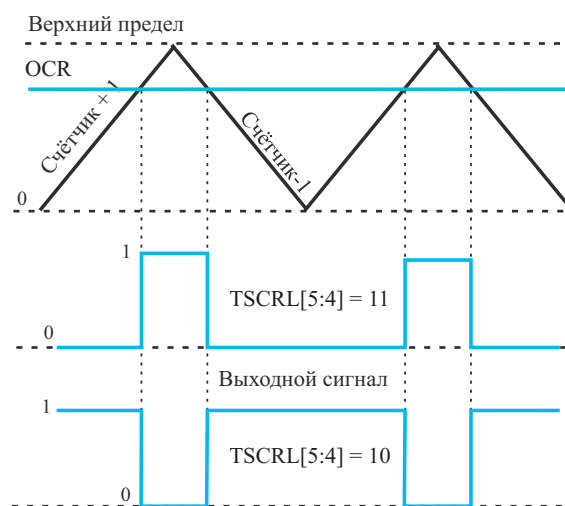


Рисунок 12. ШИМ с фазовой коррекцией

3.5.6. Двухканальный цифровой осциллограф

Цифровой осциллограф представляет собой устройство, выводящее поступающие на его каналы цифровые сигналы в виде графиков. Устройство содержит два аналогичных друг другу канала. Описываемая версия устройства предназначена для отображения состояния выходной линии Таймера-5 в различных режимах его работы.

Учитывая, что в системе fN8 допускается подключение нескольких экземпляров ВУ одного типа, но с разными базовыми адресами, два канала прибора можно подключать к разным Таймерам-5 и сравнивать их поведение.

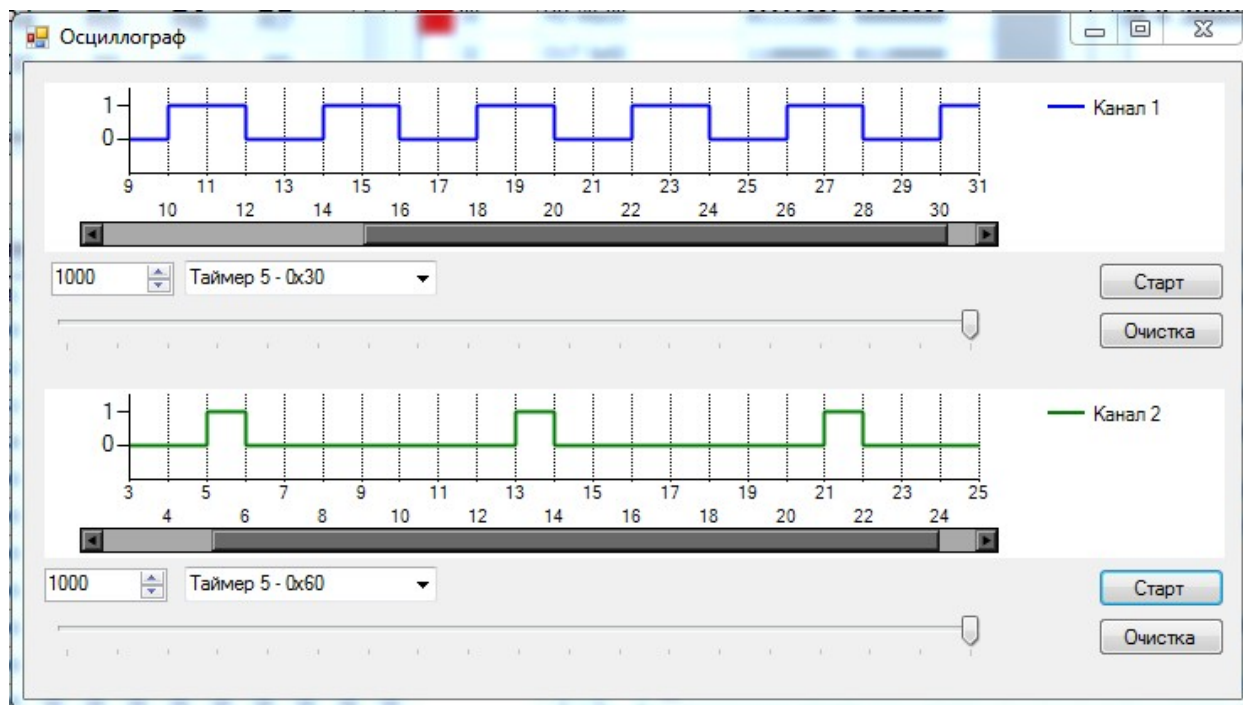


Рисунок 13. Экран цифрового осциллографа

Важно, чтобы подключение осциллографа осуществлялось **после** того, как к системе подключили таймеры. В этом случае программная модель «Осциллограф» формирует список подключённых к системе Таймеров-5 с их базовыми адресами и предоставляет возможность выбора таймера, подключаемого к каждому каналу.

Частота опроса состояния вывода Таймера-5 по каждому каналу выбирается независимо (в диапазоне от 100 мс до 1000 мс) с помощью поля ввода или «ползунка».

Кнопка *Старт/Стоп* запускает фиксацию состояний вывода таймера и вывод графика на экран. Повторное нажатие останавливает этот процесс. Однако зафиксированные значения сохраняются в памяти и допускается их просмотр. Кнопка *Очистка* обеспечивает сброс памяти канала и удаление изображения графика с экрана.

3.6. Контроллер матричной клавиатуры и 7-сегментной динамической индикации

Контроллер предназначен для управления динамической 7-сегментной индикацией и матричной клавиатурой.

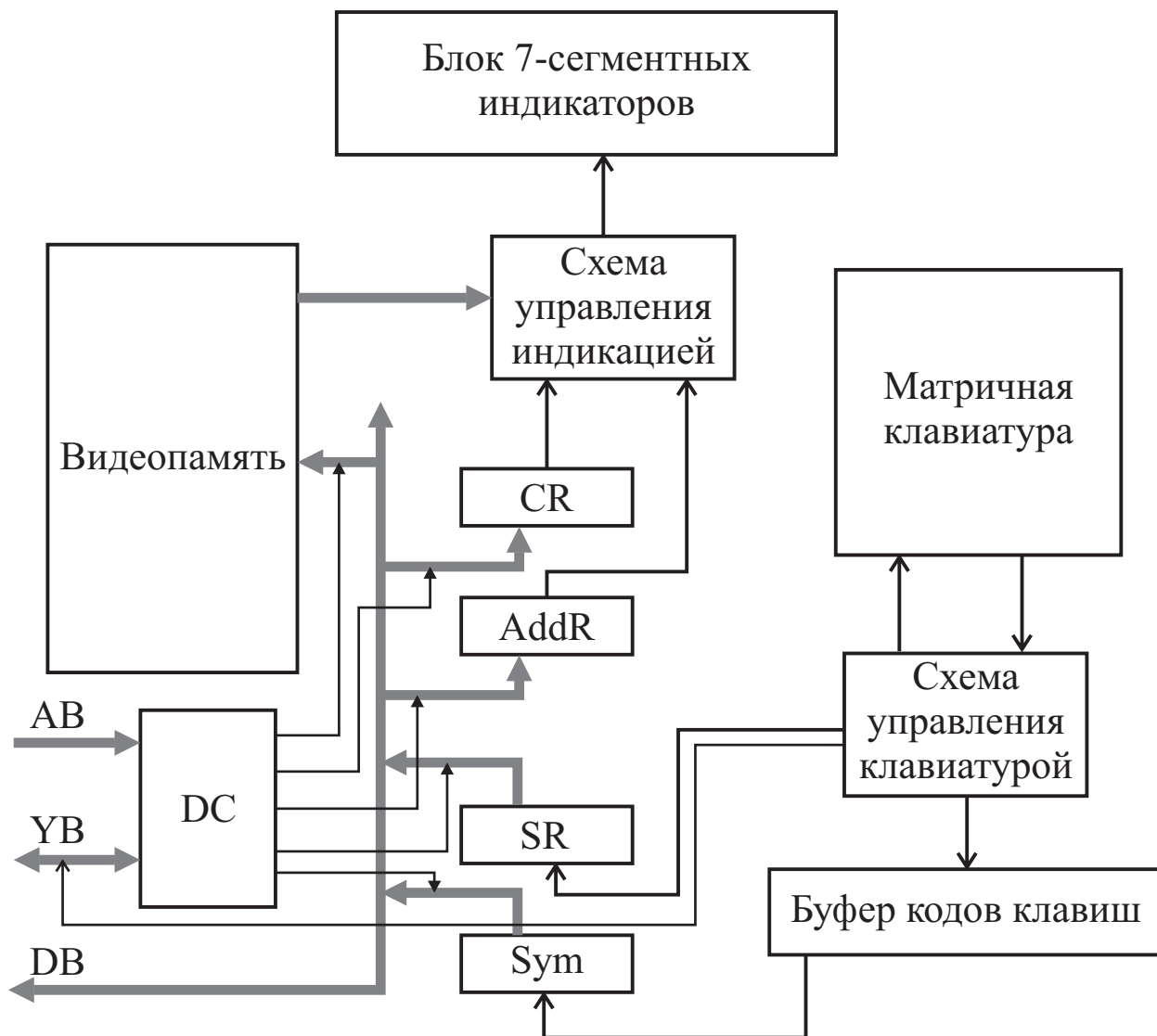


Рисунок 14. Функциональная схема контроллера клавиатуры и индикации

Функциональная схема контроллера представлена на рис. 14. Он состоит из двух слабо связанных между собой частей:

- 1) схем, управляющих выводом символов на блок 7-сегментных индикаторов:
 - видеопамять, хранящая 7-сегментные коды отображаемых символов;
 - программно-доступный регистр адреса видеопамати Addr;
 - программно-доступный регистр управления CR;
 - логических схем управления индикацией;
- 2) схем, управляющих сканированием клавиш, определением кода нажатой клавиши, сохранением кодов в буфере и информированием процессора о событии нажатия клавиши:
 - логические схемы управления модулем клавиатуры;
 - буфер кодов нажатых клавиш («очередь»);
 - программно-доступный регистр состояния контроллера SR;
 - программно-доступный регистр Sum кода клавиши (вершина буфера).

Контроллер связан с процессором по системной магистрали, включающей шину данных *DB* (8 бит), шину адреса *AB* (7 бит)⁸ и линии шины управления *YB*: *RDIO* – чтение из регистра ввода/вывода, *WRIO* – запись в регистр ввода/вывода.

К контроллеру можно подключать блоки 7-сегментных индикаторов и матрицы клавиатуры различной размерности. Настройки контроллера позволяют выбрать размерность матрицы клавиатуры 3×3 , 3×4 , 4×4 и блоки индикаторов на 2, 4, 6, или 8 разрядов.

Допускается работа контроллера только с модулем индикации, без подключения клавиатуры.

Настройки позволяют установить положение точки (сегмент Н) в формате 7-сегментного кода символа: GFEDCBAH (по умолчанию) или HGFEDCBA.

Окно обозревателя устройства в режиме «8 индикаторов и клавиатура 4×4 » показано на рис. 16.

В адресном пространстве ввода/вывода контроллер использует пять адресов:

- 0 – регистр адреса видеопамати Addr. Значение этого регистра всегда устанавливается по модулю выбранной разрядности индикации.
- 1 – регистр кода клавиши Sum отображает двоичный код первого элемента буфера (очереди введённых символов клавиатуры). Код клавиши соответствует двоичному коду цифры, на ней изображенной, причём «*» ~ E (1110), а «#» ~ F (1111).

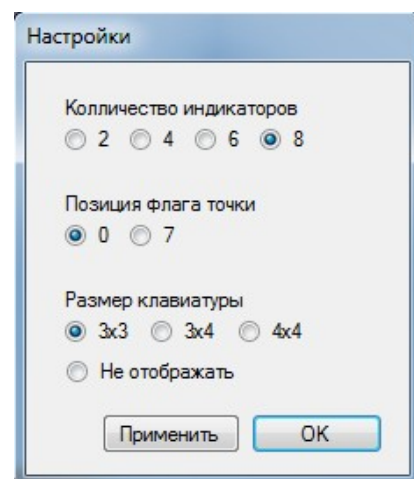


Рисунок 15. Окно настройки

⁸Шина адреса *AB* системной магистрали fN8 имеет 10 разрядов, внешние устройства используют только 7 младших из них, причём 4 младших разряда *AB*[3 : 0] используются для выбора регистра в составе ВУ, а *AB*[6 : 4] – номер ВУ (базовый адрес: *AB*[6 : 4].0000).

2 – регистр управления CR содержит три флага управления режимами работы контроллера:

CR[0] = E – флаг включения контроллера;

CR[1] = A – флаг разрешения автоинкремента Addr (по модулю выбранной разрядности индикации!) после каждой записи в видеопамять;

CR[2] = I – флаг разрешения формирования запроса на прерывание при не пустом буфере клавиатуры.

3 – регистр состояния SR содержит информацию о состоянии буфера клавиатуры:

SR[3:0] – количество находящихся в буфере не считанных кодов клавиш. Учитывая, что размер буфера равен 8, при SR[3] = 1 считается, что буфер заполнен, дальнейшая запись в него блокируется и коды нажимаемых клавиш теряются. При считывании из буфера (из регистра Sym) очередного символа значение SR[3:0] декрементируется и запись в буфер возобновляется;

SR[7] – флаг «Буфер пуст», устанавливается в «1», когда в буфере отсутствуют коды клавиш (SR[3:0] = 0000).

4 – регистр данных видеопамати. Этот регистр является виртуальным, фактически запись (OUT) производится в ячейку видеопамати по адресу, установленному в Addr. Чтение (IN) из этого регистра возвращает 0.

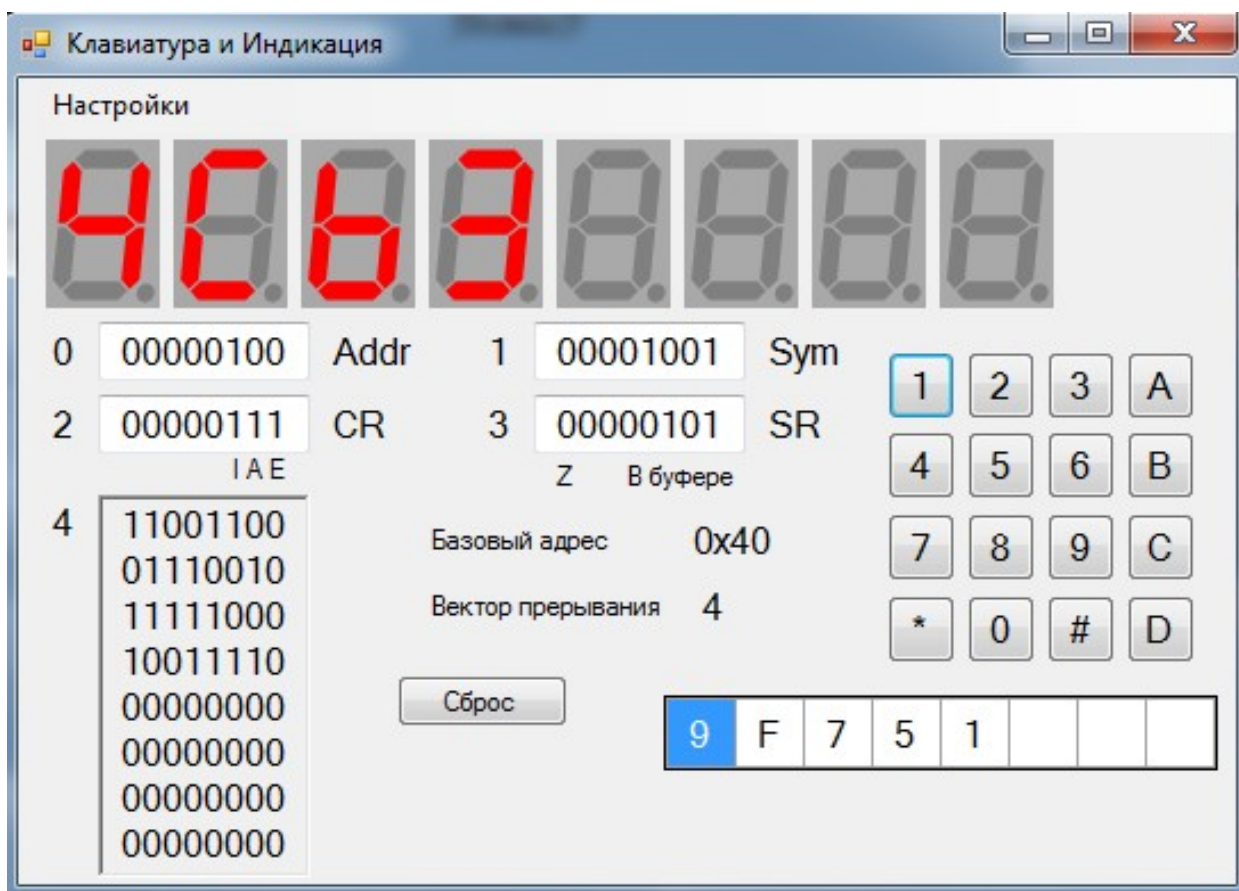


Рисунок 16. Окно обозревателя контроллера клавиатуры и индикации

Нажатие в окне обозревателя кнопки *Сброс* очищает видеопамять (и соответственно, гасит индикацию), буфер клавиатуры и все регистры. Те же действия можно реализовать программно, если записать в CR любой байт с «1» в старшем разряде или выполнить команду SBI 0x42,7.

При непустом буфере и установленном в CR флаге $I = 1$ контроллер формирует запрос на прерывание (по умолчанию – вектор 4). Запрос снимается автоматически, когда буфер становится пуст.

3.7. Цветной графический дисплей

Программная модель цветного графического дисплея (CGD) предназначена для подключения в качестве внешнего устройства к программной модели процессорного ядра fN8.

Основные параметры:

- Размер графического экрана – 128×64 пиксела модели.
- Размер пикселя модели - 4×4 пиксела экрана.
- Количество цветов текущей палитры – 16.
- Объём видеопамати – 4096 байта (8192 пиксела).
- Метод формирования цвета – ARGB (4 байта на цвет).
- Объём памяти палитры – 64 байта.

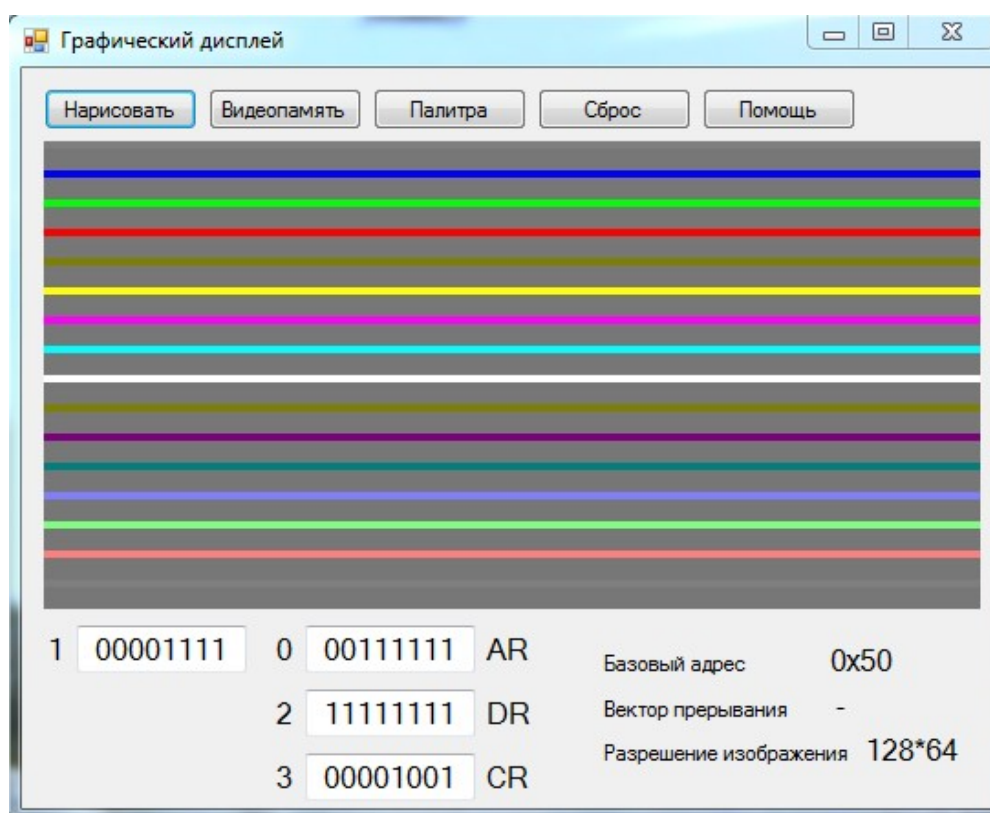


Рисунок 17. Окно CGD с рисунком горизонтальных линий всех цветов текущей палитры

Изображение на экране CGD (128×64 пиксела) определяется содержимым видеопамати и памяти палитры. Дамп видеопамати (16×256 байт) открывается в специальном окне (рис. 18) по нажатию кнопки *Видеопамать*. Каждый байт видеопамати содержит номера цветов текущей палитры для двух соседних пикселей, причем младшая тетрада байта соответствует пикселу с меньшим номером.

Соотношение между номером пиксела P (в диапазоне $[0 \dots 8191]$) и адресом видеопамяти A (в диапазоне $[0 \dots 4095]$) определяется выражениями:

$$A = P \operatorname{div} 2; \quad t = P \operatorname{mod} 2,$$

где t – номер тетрады в байте: 0 – младшая, 1 – старшая.

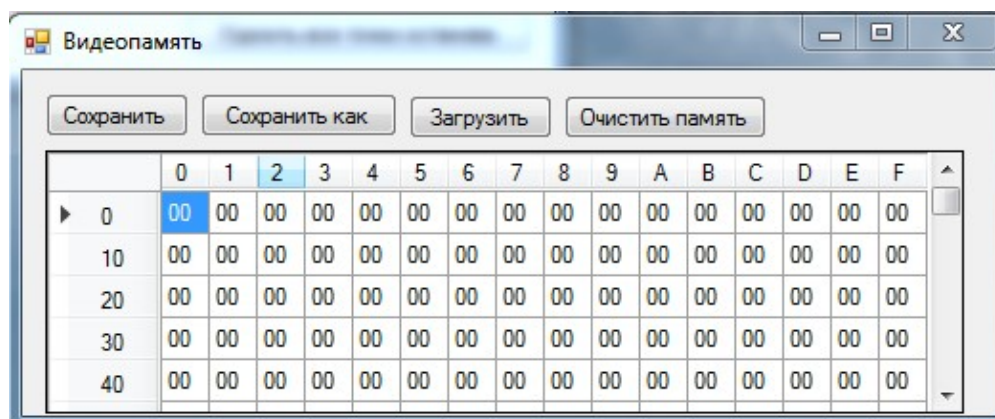


Рисунок 18. Фрагмент окна видеопамяти

На экране модели CGD координаты пиксела (x, y) , где x – номер столбца, y – номер строки, связаны с адресом видеопамяти A следующими соотношениями:

$$x = 2 \cdot (A \operatorname{mod} 64) + t; \quad y = A \operatorname{div} 64.$$

Пиксел с координатами $(0, 0)$ расположен в *левом верхнем* углу экрана CGD. Очевидно, что $P = 128y + x$.

Текущая палитра загружается при подключении CGD к fN8 и выводится в специальное окно (рис. 19) по нажатию кнопки *Палитра* главного окна. Она включает 16 цветов, причём цвет под номером 0 всегда используется как цвет фона экрана. Каждый цвет задаётся четвёркой байт, первый из которых определяет уровень насыщенности цвета, а остальные три – уровни базовых цветов RGB (красный, зелёный, синий). Все уровни задаются целыми шестнадцатеричными числами от 0x00 до 0xFF.

Видеопамять и память палитры допускают редактирование содержимого в «ручном режиме» путем изменения значений в ячейках. Содержимое видеопамяти и/или палитры можно сохранять в виде файлов, а так же загружать из файлов. Для выполнения таких действий в окнах *Видеопамять* и *Память палитры* предусмотрены соответствующие кнопки.

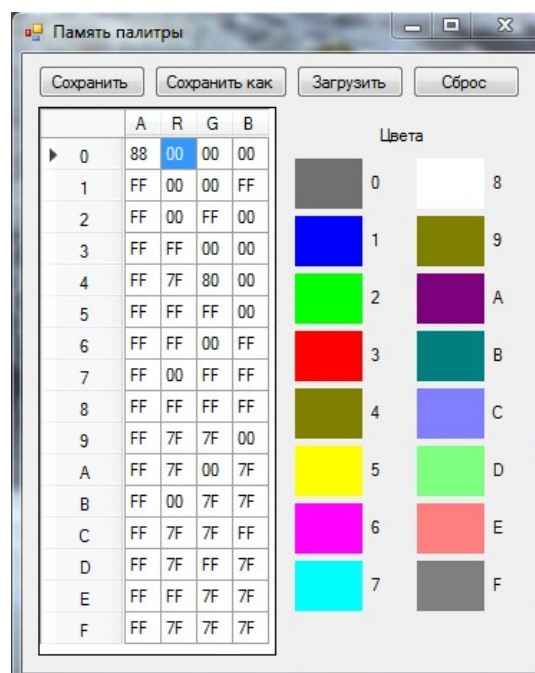


Рисунок 19. Палитра

Для программного доступа в видеопамять и редактирования палитры используются четыре программно-доступных регистра:

0 – регистр ARL младшего байта адреса;

1 – регистр ARH старшего байта адреса;

2 – регистр DR данных;

3 – регистр CR управления:

CR[0] – включение устройства;

CR[1] – автоинкремент AR после обращения к DR;

CR[2] – выбор типа памяти: 0 – видеопамять, 1 – память палитры;

CR[3] – обновление экрана: 0 – по нажатию кнопки «Перерисовать», 1 – сразу после изменения в памяти.

Регистры AR и DR реализуют окно интерфейса между адресным пространством ввода/вывода fN8 и блоками памяти CGD. Команда IN 0x52 считывает в Асс содержимое ячейки, адрес которой установлен в AR, тип памяти (видео или палитра) выбирается в соответствии с текущим значением CR[2], причём адрес формируется по модулю объёма выбранной памяти (4096 или 64), а значение лишних старших разрядов AR игнорируется. Аналогично производится загрузка ячеек по командам OUT 0x52. При этом, если установлен флаг CR[1]=1, то каждое обращение к DR сопровождается последующим инкрементом AR.

Часть II

Лабораторный практикум

4. Лабораторная работа № 1.1. Знакомство с моделью. Программирование простого разветвляющегося процесса

В данной работе необходимо разработать и отладить на модели fN8 программу, вычисляющую значение простой функции $F_N(x)$, если значение x попадает в заданный для этой функции допустимый диапазон аргумента. Программа должна получать на входе значение аргумента и выводить значение функции, соответствующее этому аргументу.

В этой работе мы не используем устройства ввода/вывода ЭВМ, поэтому будем осуществлять ввод и вывод непосредственно через ячейки оперативной памяти.

ЭВМ fN8 имеет длину слова 8 бит. Будем рассматривать формат данных в этой работе как **целое без знака**. Тогда диапазон чисел, с которыми может оперировать программа, составляет $[0..255]$.

Если входной аргумент x выходит за пределы заданного диапазона или результат любой операции при вычислении функции выходит за пределы $[0..255]$ следует сформировать сообщение об ошибке и завершить программу. Поскольку значение вычисляемой функции теоретически может принять любое из 256 значений байта, то ни одно из них нельзя использовать как признак ошибки. Поэтому следует использовать для вывода результата две ячейки памяти: в одну выводится вычисленное значение функции, а в другую – признак ошибки (например, FF) или её отсутствия (например, 00).

Внимание! Следует помнить, что числа в памяти fN8 хранятся, вводятся и выводятся **в шестнадцатеричной системе счисления**. При написании программы на Ассемблере можно использовать десятичные (39), шестнадцатеричные (0x27) и даже двоичные (0b100111) константы, но в память они помещаются (и отображаются) только в шестнадцатеричном формате (в этих примерах, как 27).

4.1. Порядок выполнения работы

1. Разработать на языке Ассемблера для fN8 текст программы, обеспечивающий решение своего варианта задания (таблица 2).
2. Ввести текст программы в окно *Компилятор*, при этом возможен набор и редактирование текста непосредственно в окне *Компилятор* или загрузка текста из файла, подготовленного в другом редакторе.
3. Ассемблировать текст программы, при необходимости исправить синтаксические ошибки.
4. Отладить программу. Для этого:
 - а) Записать в выбранную для ввода аргумента значение x (в области допустимых значений!).

- б) Проверить правильность выполнения программы (т.е. правильность результата и адреса останова) в режиме Автомат (кнопка *Запуск*). В случае обнаружения ошибки выполнить *Сброс* процессора модели (PCL на точку старта), выполнить программу в режиме *Шаг* и, наблюдая результаты выполнения каждой команды, найти команду, являющуюся причиной ошибки.
- с) С помощью отлаженной программы получить результаты работы программы⁹ для 3-4 значений аргумента внутри допустимого диапазона, на его границах (2 значения) и за пределами диапазона (2 значения).

4.2. Содержание отчета

1. Формулировка варианта задания.
2. Граф-схема алгоритма решения задачи.
3. Размещение данных и результата в ОЗУ.
4. Текст программы на языке Ассемблера fN8 с комментариями.
5. Таблица результатов работы программы для нескольких значений аргумента.

4.3. Контрольные вопросы

1. Дайте определение понятия «Команда (инструкция) ЭВМ».
2. Какие форматы команд Вам известны?
3. Что такое «командный цикл процессора»?
4. Как записать программу в машинных кодах в память модели ЭВМ?
5. Как просмотреть содержимое регистров процессора и изменить содержимое некоторых регистров?
6. Как просмотреть и, при необходимости, отредактировать содержимое ячейки памяти?
7. Как запустить выполнение программы в режиме приостановки работы после выполнения каждой команды?
8. Какие способы адресации операндов применяются в командах ЭВМ?
9. Какие команды относятся к классу команд передачи управления?
10. Как работают команды передачи управления? Содержимое каких регистров процессора может изменяться при их выполнении?
11. Чем отличается работа команд условных переходов от команды JMP xxx?
12. Какие флаги могут анализироваться при выполнении команд условных переходов?
13. Будет ли осуществлена передача управления на метку M1 командой JN M1, если $Z = 1$?

⁹Результат работы программы – содержимое двух ячеек ОЗУ: значение функции и код ошибки. При наличии ошибки значение функции не определено.

14. Как определить, одинаковы ли значения, хранящиеся в двух заданных ячейках ОЗУ?
15. Что входит в понятие «отладка программы»?
16. Какие способы отладки программы можно реализовать в программной модели учебной ЭВМ?

4.4. Варианты задания

Таблица 2. Варианты задания

N	$F(x)$	N	$F(x)$
1	$\frac{x^2 + 17}{1 + x}; \quad 2 \leq x \leq 8$	2	$4x^2 - 5x + 3; \quad 0 \leq x \leq 18$
3	$\frac{58 - x^2}{15}; \quad 0 \leq x \leq 9$	4	$\frac{11 + 8x}{14 - x}; \quad 2 \leq x \leq 15$
5	$\frac{233 - x}{4x^2 - 2x}; \quad 5 \leq x \leq 9$	6	$\frac{2x^2 - 5x + 3}{11}; \quad 3 \leq x \leq 11$
7	$\frac{133 - 2x^2}{3x - 2}; \quad 5 \leq x \leq 8$	8	$\frac{151 - 14x}{13 - x}; \quad 4 \leq x \leq 12$
9	$\frac{3x^2 - x + 8}{3(x - 1)}; \quad 4 \leq x \leq 11$	10	$x^3 - 2x^2 + 3x - 4; \quad 5 \leq x \leq 8$
11	$\frac{x^3 - 28}{x - 1}; \quad 2 \leq x \leq 7$	12	$5 + \frac{2x^2 - 11x + 2}{4}; \quad 5 \leq x \leq 10$
13	$\frac{x^3 - 2x^2 + 3x - 4}{3}; \quad 6 \leq x \leq 12$	14	$\frac{45x + 7}{12 - x} - 12; \quad 2 \leq x \leq 9$
15	$\frac{85 + 4x - x^2}{x + 8}; \quad 1 \leq x \leq 7$	16	$\frac{x^3 + 3x^2 - 2}{2} + 4; \quad 1 \leq x \leq 6$

5. Лабораторная работа № 1.2

Программирование цикла

При решении задач, связанных с обработкой массивов, возникает необходимость изменения исполнительного адреса при повторном выполнении некоторых команд. Эта задача может быть решена путем использования косвенной адресации.

5.1. Пример 2

Разработать программу вычисления суммы элементов массива чисел C_1, C_2, \dots, C_n . Исходными данными в этой задаче являются: n — количество суммируемых чисел и C_1, C_2, \dots, C_n — массив суммируемых чисел. Заметим, что должно выполняться условие $n > 1$, т.к. алгоритм предусматривает по крайней мере одно суммирование. Кроме того, предполагается, что суммируемые числа записаны в ОЗУ подряд, т.е. в ячейки памяти с последовательными адресами. Результатом является сумма S .

Составим программу для вычисления суммы со следующими конкретными параметрами: число элементов массива — 6, элементы массива расположены в ячейках сегмента данных ОЗУ по адресам 0x40, 0x41, 0x42, ..., 0x45. Используем в качестве рабочих ячеек регистры R0 — текущий адрес массива, R1 — размер массива, R2 — текущая сумма.

В системе команд fN8 реализованы разновидности косвенно-регистровой адресации, включая постинкрементную, которой в нашем случае удобно воспользоваться.

Текст программы Примера 2

1	RD #0x40	; загружаем адрес начала массива
2	WR R0	; в регистр R0
3	RD #6	; размер массива —
4	WR R1	; в R1 (счётчик цикла)
5	RD #0	; обнуляем начальное значение суммы
6	WR R2	; в R2
7	M1: RD R2	; загружаем в Асс текущее значение суммы
8	ADD @R0+	; суммируем очередной элемент и переходим к след. адресу
9	WR R2	; сохраняем в R2 новое значение суммы
10	DJRNZ R1,M1	; декремент счётчика и переход на начало цикла, если R1 $\neq 0$
11	HLT	; стоп
12		;
13	.c 2	; устанавливается сегмент данных в качестве текущего
14	.org 0x40	; устанавливается текущий адрес компиляции
15	.db 12, 7, 21	; загрузка байтов по текущим адресам
16	.db 4, 11, 117	; то же (продолжение)

Для отладки приведённой программы следует записать в ячейки 0x40...0x45 произвольные числа¹⁰. Загрузку в ячейки сегмента данных можно выполнить вручную или поручить её компилятору. Строки 13 – 16 примера содержат директивы компилятору (см. раздел 1.3), обеспечивающие эту загрузку, причём строки 15 и 16 можно объединить: `.db 12, 7, 21, 4, 11, 117`

Следует учитывать, что в приведённом примере мы используем десятичные числа, но компилятор разместит в памяти их шестнадцатеричные эквиваленты и после компиляции можно будет увидеть в ячейках 0x40...0x45 сегмента данных значения 0C, 07, 15, 04, 0B, 75. Процессор работает так же в двоичной (шестнадцатеричной) системе и после завершения работы программы в регистре R2 получим AC. ($AC_{16} = 172_{10}$).

5.2. Задание 2

Написать и отладить программу определения заданной характеристики последовательности чисел C_1, C_2, \dots, C_n . Варианты заданий приведены в таблице 3.

Таблица 3. Варианты задания 2

№ вар.	Характеристика последовательности чисел C_1, C_2, \dots, C_n
1	Количество чётных чисел
2	Номер минимального числа
3	Минимальное число из пары <сумма чётных> и <сумма нечётных> чисел в массиве
4	Номер первого отрицательного числа
5	Количество чисел, равных C_1
6	Количество отрицательных чисел
7	Максимальное отрицательное число
8	Номер первого положительного числа
9	Минимальное положительное число
10	Номер максимального числа
11	Количество нечётных чисел
12	Количество чисел, меньших C_1
13	Разность сумм чётных и нечетных элементов массива
14	Максимальное число из пары <сумма чётных> и <сумма нечётных> элементов массива
15	Сумма элементов массива кратных 3
16	Сумма элементов массива кратных 11

Примечание. Под чётными (нечётными) элементами массива здесь понимаются элементы массивов, имеющие чётные (нечётные) индексы. Чётные числа — элементы массивов, делящиеся без остатка на 2.

¹⁰Следует помнить, что если $S > 255$, то приведённая программа сформирует ответ как $(S \bmod 256)$.

5.3. Содержание отчета

1. Формулировка варианта задания.
2. Граф-схема алгоритма решения задачи.
3. Распределение памяти (размещение в регистрах и ОЗУ переменных, программы и необходимых констант).
4. Текст программа.
5. Тестовые примеры и результаты их работы.

5.4. Контрольные вопросы

1. Как организовать цикл в программе?
2. Что такое «параметр цикла»?
3. Как поведет себя программа, приведенная на стр.43, если в ней в строке 9 вместо команды **ADD @R0+** окажется команда **ADD @R0?** **ADD @R0-**? **ADD +@R0?**
4. Как поведет себя программа, приведенная на стр.43, если метка M1 будет поставлена в начале строки 006? в начале строки 008?

6. Лабораторная работа № 1.3

Подпрограммы и стек

В программировании часто встречаются ситуации, когда одинаковые действия необходимо выполнять многократно в разных частях программы (например, вычисление функции $\sin x$). При этом с целью экономии памяти не следует многократно повторять одну и ту же последовательность команд – достаточно один раз написать так называемую *подпрограмму* (в терминах языков высокого уровня – процедуру) и обеспечить правильный вызов этой подпрограммы и возврат в точку вызова по завершению подпрограммы.

Для вызова подпрограммы необходимо указать ее начальный адрес в памяти и передать ей (если необходимо) параметры – те исходные данные, с которыми будут выполняться предусмотренные в подпрограмме действия. Адрес подпрограммы указывается в команде вызова *CALL*, а параметры могут передаваться через определенные ячейки памяти, регистры или стек.

Возврат в точку вызова обеспечивается сохранением адреса текущей команды (содержимого *PC*) при вызове и использованием в конце подпрограммы команды возврата *RET*, которая возвращает сохраненное значение адреса возврата в *PC*.

Для реализации механизма вложенных подпрограмм (возможность вызова подпрограммы из другой подпрограммы и т.д.) адреса возврата целесообразно сохранять в стеке. *Стек* («магазин») – особым образом организованная безадресная память, доступ к которой осуществляется через единственную ячейку, называемую *верхушкой стека*. При записи слово помещается в верхушку стека, предварительно все находящиеся в нем слова смещаются «вниз» на одну позицию; при чтении извлекается содержимое верхушки стека (оно при этом из стека исчезает), а все оставшиеся слова смещаются «вверх» на одну позицию. Такой механизм напоминает действие магазина стрелкового оружия (отсюда и второе название). В программировании называют такую дисциплину обслуживания LIFO (Last In First Out) – «последним пришел, первым вышел» – в отличие от дисциплины типа «очередь» FIFO (First In First Out).

В обычных ОЗУ нет возможности перемещать слова между ячейками, поэтому при организации стека перемещается не массив слов относительно неподвижной верхушки, а верхушка относительно неподвижного массива. Под стек отводится некоторая область ОЗУ, причем адрес верхушки хранится в специальном регистре процессора – *указателе стека SP*. В стек можно поместить содержимое регистра общего назначения по команде *PUSH* или извлечь содержимое верхушки в регистр общего назначения по команде *POP*. Кроме того, по команде вызова подпрограммы *CALL* значение программного счетчика *PC* (адрес следующей команды) помещается в верхушку стека, а по команде *RET* содержимое верхушки стека извлекается в *PC*. При каждом обращении в стек указатель *SP* автоматически модифицируется.

В большинстве ЭВМ стек «растет» в сторону меньших адресов, поэтому перед каждой записью в стек содержимое *SP* уменьшается на 1, а после каждого извлечения из стека содержимое *SP* увеличивается на 1. Таким образом *SP* всегда указывает на верхушку стека.

Цель настоящей лабораторной работы – изучение организации программ с использованием подпрограмм.

6.1. Пример 3

Даны три массива чисел. Требуется вычислить среднее арифметическое их максимальных элементов. Каждый массив задается двумя параметрами: адресом первого элемента и длиной.

Очевидно, в программе трижды необходимо выполнить поиск максимального элемента массива, поэтому следует написать соответствующую подпрограмму.

Параметры в подпрограмму будем передавать через регистры: R_0 — начальный адрес массива, R_1 — длина массива.

Рассмотрим конкретную реализацию этой задачи. Пусть первый массив начинается с адреса $0x10$ и имеет длину 6 элементов, второй — $0x20$ и 7, третий — $0x30$ и 8. Программа будет состоять из основной части и подпрограммы. Основная программа трижды задает параметры массивов подпрограмме, вызывает ее и сохраняет результаты работы подпрограммы в рабочих ячейках. Затем осуществляет вычисление среднего арифметического и результат программа оставляет в Асс.

В качестве рабочих ячеек используются регистры общего назначения R_5 и R_4 — для хранения максимальных элементов массивов. Подпрограмма получает параметры через регистры R_0 (начальный адрес массива) и R_1 (длина массива). Эти регистры используются подпрограммой в качестве регистра текущего адреса и счетчика цикла соответственно. Кроме того, R_2 используется для хранения текущего максимума и по завершению цикла подпрограммы в R_2 остаётся результат — максимальный элемент массива. Ниже (стр. 49) приведен текст основной программы и подпрограммы.

6.2. Задание 3

Составить и отладить программу учебной ЭВМ для решения следующей задачи. Три массива в памяти заданы начальными адресами и длинами. Вычислить и вывести на устройство вывода среднее арифметическое параметров этих массивов. Параметры определяются заданием к предыдущей лабораторной работе (таблица 3 на стр. 44), причем соответствие между номерами вариантов заданий 2 и 3 устанавливается таблицей 4.

Таблица 4.

№ варианта задания 3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
№ строки в табл 3	5	7	13	16	9	12	1	10	14	2	6	8	15	4	11	3

6.3. Содержание отчета

1. Формулировка варианта задания.
2. Граф-схема алгоритма основной программы.
3. Граф-схема алгоритма подпрограммы.

4. Распределение памяти (размещение в ОЗУ переменных, программы и необходимых констант).
5. Текст программы и подпрограммы.
6. Значения исходных данных и результата выполнения программы.

Текст программы Примера 3

```

1   ;    Основная программа
2   RD #0x10          ; загружаем адрес начала 1-го массива
3   WR R0              ; в регистр R0
4   RD #6              ; размер 1-го массива –
5   WR R1              ; в R1 (счётчик цикла)
6   CALL MX            ; вызов подпрограммы MX
7   MOV R5,R2          ; сохраняем  $C_{max}1$  в R5
8   RD #0x20           ; загружаем адрес начала 2-го массива
9   WR R0              ; в регистр R0
10  RD #7              ; размер 2-го массива –
11  WR R1              ; в R1 (счётчик цикла)
12  CALL MX            ; вызов подпрограммы MX
13  MOV R4,R2          ; сохраняем  $C_{max}2$  в 4
14  RD #0x30           ; загружаем адрес начала 3-го массива
15  WR R0              ; в регистр R0
16  RD #8              ; размер 3-го массива –
17  WR R1              ; в R1 (счётчик цикла)
18  CALL MX            ; вызов подпрограммы MX
19  RD R2              ; загружаем в Асс  $C_{max}3$ 
20  ADD R5              ; добавляем  $C_{max}1$ 
21  ADD R4              ; добавляем  $C_{max}2$ 
22  DIV #3             ; вычисляем  $C_{max}$ Среднее
23  HLT                ;
24  ;    Подпрограмма
25  MX:              ;
26  RD @R0             ; читаем 1-й элемент массива и
27  WR R2              ; загружаем в R2 как текущий максимум
28  M1: RD @R0+       ; читаем очередной элемент и переход к след. адресу
29  CMP R2              ; сравниваем с текущим максимумом ( $R2 - C_i$ )
30  JNN M2              ; если разность не отрицательна – не меняем R2
31  WR R2              ; записываем в R2 текущий элемент массива  $C_i$ 
32  M2: DJRNZ R1,M1   ; меняем параметр цикла и проверяем завершение
33  RET                ; возврат в вызывающую программу

```

Для тестирования программы требуется разместить в памяти элементы массивов (здесь тоже рассматриваются как целые без знака в диапазоне $[0..0xFF]$). Можно произвести загрузку ячеек вручную, но удобнее использовать директивы компилятора. Например, массивы для тестирования задачи примера 3 можно создать таким образом:

35	<code>.c 2</code>	; переходим в сегмент данных
36	<code>.org 0x10</code>	; начальный адрес 1-го массива
37	<code>.db 12, 7, 21, 4, 11, 117</code>	; 6 элементов 1-го массива
38	<code>.org 0x20</code>	; начальный адрес 2-го массива
39	<code>.db 2, 0, 11, 3, 1, 88, 21</code>	; 7 элементов 2-го массива
40	<code>.org 0x30</code>	; начальный адрес 3-го массива
41	<code>.db 15, 46, 13, 6, 13, 37, 44, 29</code>	; 8 элементов 3-го массива

Здесь элементы массивов заданы десятичными числами, но в ячейках памяти они разместятся как их шестнадцатеричные эквиваленты.

6.4. Контрольные вопросы

1. Какие способы адресации РОН используются в программной модели учебной ЭВМ?
2. Как работает команда *MOV R3, R7*?
3. Какие действия выполняет процессор при реализации команды *CALL x*?
4. Как поведет себя программа, отлаженная Вами в рамках выполнения лабораторной работы №3, если
 - подпрограмму завершить двумя подряд стоящими командами *RET*?
 - вместо одной поставить две подряд стоящие команды *CALL x* после задания подпрограмме параметров 1-го массива?
 - перейти к подпрограмме не командой *CALL x*, а командой *JMP x*?
5. После начальной установки процессора (сигнал Сброс) указатель стека *SPL* устанавливается в 00. По какому адресу будет производится запись в стек первый раз, если не загружать *SPL* командой *MOVSP*?
6. Как, используя механизмы постинкрементной и преддекрементной адресации, организовать дополнительный стек пользователя в произвольной области памяти, не связанный с *SP*?

7. Лабораторная работа № 1.4

Программирование внешних устройств

Целью этой лабораторной работы является изучение способов организации взаимодействия процессора и внешних устройств (ВУ) в составе ЭВМ.

В описываемой версии модели fN8 набор ВУ включает два дисплея (символьный и графический), блок управления клавиатурой, контроллер матричной клавиатуры и 7-сегментной индикации и две таймерные системы (*Таймер 2* и *Таймер 5*). С помощью механизма *Диспетчер внешних устройств* можно подключать к *Таймеру 5* двухканальный цифровой осциллограф, который ВУ ЭВМ, собственно, не является и предназначен для отображения выходного сигнала *Таймера 5*.

Функционирование перечисленных ВУ подробно описано в разделе 3 настоящего пособия.

Связь процессора и ВУ может осуществляться в синхронном или асинхронном режиме. Синхронный режим используется для ВУ, всегда готовых к обмену. В нашей модели такими ВУ являются оба дисплея.

Остальные ВУ могут формировать события, контролируемые системой: нажатие клавиши, переполнение буфера, совпадение состояния счётчика с заданной константой, переполнение счётчика и др.

Обнаружение факта возникновения события может осуществляться процессором двумя способами:

1. В программно-управляемом режиме путём анализа флагов в регистрах ВУ.
2. В режиме прерывания (см. раздел 1.5).

В первом случае предполагается программное обращение процессора к регистру состояния ВУ с последующим анализом значения соответствующего разряда слова состояния. Такое обращение следует предусмотреть в программе с некоторой периодичностью, независимо от фактического наступления контролируемого события (например, нажатие клавиши на клавиатуре).

Во втором случае при возникновении контролируемого события ВУ формирует процессору запрос на прерывание программы, по которому процессор и осуществляет связь с ВУ.

7.1. Задания

В работе будем использовать, в зависимости от варианта задания, несколько ВУ из множества, входящего в набор ВУ fN8. Для краткости введём для них следующие обозначения:

- Klav – Контроллер клавиатуры (см. раздел 3.2)
- SDisp – Символьный дисплей (см. раздел 3.3)
- 7Seg – Контроллер матричной клавиатуры и 7-сегментной индикации (см. 3.6)
- T2 – Таймер-2 (см. раздел 3.4)
- T5 – Таймер-5 (см. раздел 3.5)

Подключение нужных ВУ осуществляется через диспетчер ВУ (см. раздел 3.1).

Таблица 5 — Варианты заданий

№	Задания	ВУ
1	Вывод на дисплей одного из трёх текстовых сообщений, в зависимости от нажатой клавиши. «1» – вывод первого текста, «2» – второго, «3» – третьего, на остальные символы реакции нет.	Klav, SDisp
2	При каждом нажатии клавиши выводить в Асс (в двоичном коде) время, прошедшее с предыдущего нажатия.	7Seg, T5
3	Вывод на дисплей одного из трёх текстовых сообщений, в зависимости от нажатой клавиши. «1» – вывод первого текста, «2» – второго, «3» – третьего, на остальные символы реакции нет.	7Seg, SDisp
4	Ввод символов с клавиатуры и вывод на дисплей. Необходимо вывести 4 строки, в каждой из которых 8 произвольных символов. Первая строка – левое выравнивание, вторая – по центру, третья – правое, четвёртая – с пробелом после каждого символа.	Klav, SDisp
5	Ввод шестнадцатеричных цифр с клавиш 7Seg и вывод их на 7-сегментную индикацию со сдвигом вправо.	7Seg
6	Вывод на дисплей цифр в верхние 4 строки, кириллицу – в нижние 4 строки, латиницу и другие символы игнорировать.	Klav, SDisp
7	Выделять из потока вводимых с клавиатуры символов только десятичные цифры и выводить их на 7-сегментную индикацию слева-направо циклически.	Klav, 7Seg
8	Ввод шестнадцатеричных цифр с клавиш 7Seg и вывод их на 7-сегментную индикацию со сдвигом влево.	7Seg
9	Выделять из потока вводимых с клавиатуры символов только десятичные цифры и выводить их на 7-сегментную индикацию со сдвигом влево (как в калькуляторе).	Klav, 7Seg
10	При каждом нажатии клавиши выводить в Асс (в двоичном коде) время, прошедшее с предыдущего нажатия.	Klav, T5
11	Выделять из потока вводимых с клавиатуры символов только десятичные цифры и выводить их на 7-сегментную индикацию со сдвигом вправо.	Klav, 7Seg
12	Ввод десятичных цифр с клавиш 7Seg и вывод их на 7-сегментную индикацию со сдвигом влево.	7Seg
13	При каждом нажатии клавиши выводить в Асс (в двоичном коде) время, прошедшее с предыдущего нажатия.	Klav, T2
14	Выделять из потока вводимых с клавиатуры символов только десятичные цифры и выводить их на 7-сегментную индикацию справа-налево циклически.	Klav, 7Seg
15	При каждом нажатии клавиши выводить в Асс (в двоичном коде) время, прошедшее с предыдущего нажатия.	7Seg, T2
16	Ввод десятичных цифр с клавиш 7Seg и вывод их на 7-сегментную индикацию со сдвигом вправо.	7Seg

7.2. Порядок выполнения работы

1. Запустить программную модель ЭВМ fN8 и подключить к ней определённые в задании ВУ (*Диспетчер внешних устройств*).
2. Написать и отладить программу, предусмотренную заданием, с использованием программного анализа флагов событий, формируемых в ВУ.
3. Изменить отлаженную в п. 2 программу таким образом, чтобы процессор реагировал на события в ВУ с помощью подсистемы прерывания.

7.3. Содержание отчёта

1. Задание и постановка задачи.
2. Распределение памяти и текст программы с программным анализом флагов событий.
3. Распределение памяти и текст программы с обработчиком прерывания.

7.4. Контрольные вопросы

1. При каких условиях устанавливается и сбрасывается флаг готовности клавиатуры RDY?
2. Каким образом после сброса дисплея организовать вывод текста на него не с первой, а с четвёртой строки экрана?
3. Как определить в ВУ 7Seg наличие и количество находящихся в буфере клавиатуры кодов клавиш?
4. При каких условиях формируется запрос на прерывание ВУ 7Seg?
5. Как по команде программы зафиксировать текущее значение времени с помощью таймера T2? T5?
6. Как обнаружить факт нажатия клавиши в устройстве Klav не используя подсистему прерываний?
7. Как регулируется масштаб измеряемого времени в таймерах T2 и T5?
8. В какой области памяти fN8 могут располагаться программы – обработчики прерываний?
9. Где в fN8 располагается и как заполняется таблица векторов прерываний?
10. Чем отличается действие процессора при выполнении команды *INT x* от действий при выполнении команды *CALL x*?
11. Какая информация сохраняется в стеке при обслуживании процессором запроса на прерывание?
12. Какие изменения в работе отлаженной Вами программы произойдут, если завершить обработчик прерываний командой RET, а не IRET?

8. Лабораторная работа № 2.1.

Арифметические операции с многобайтовыми данными

8.1. Кодирование числовых данных

Многоразрядные числа в ЭВМ могут храниться (и обрабатываться) обычно представленными в шестнадцатеричной (двоичной) или в десятичной системах счисления. При этом различают два формата представления — *упакованный* и *распакованный*.

Упакованный формат позволяет хранить в одном байте две шестнадцатеричные или десятичные цифры — по одной в каждой тетраде.

Распакованный формат при хранении как правило совпадает с ASCII-кодом соответствующего символа (цифры), причем для шестнадцатеричных цифр A..F следует выбрать заглавные или строчные коды соответствующих букв.

На рис. 20 показано, как может храниться десятичное число 382 706 в разных форматах: по адресам 0x210 .. 0x215 — распакованный формат, по адресам 0x220 .. 0x222 — упакованный формат того же числа.

	0	1	2	3	4	5	6	7	8
200	00	00	00	00	00	00	00	00	00
210	36	30	37	32	38	33	00	00	00
220	06	27	38	00	00	00	00	00	00

Рисунок 20. Десятичные данные в ОЗУ

По меньшему адресу принято хранить младший фрагмент данных.

Пример форматов шестнадцатеричных данных показан на рис. 21. Восьмиразрядное шестнадцатеричное число 4E 2F CB 05 представлено в упакованном формате по адресам 0x210 .. 0x213 (его можно рассматривать как 32-разрядное двоичное число), а по адресам 0x220 .. 0x227 — распакованный формат — ASCII-коды десятичных цифр и строчных латинских букв *a..h*. Если требуется в качестве старших шестнадцатеричных цифр использовать заглавные буквы, то в старшей тетраде кода буквы следует вместо «б» поставить «4».

	0	1	2	3	4	5	6	7	8
200	00	00	00	00	00	00	00	00	00
210	05	CB	2F	4E	00	00	00	00	00
220	35	30	62	63	66	32	65	34	00

Рисунок 21. HEX-данные в ОЗУ

В арифметических операциях распакованный формат можно преобразовать в упакованный или выполнять операции непосредственно над распакованными числами с соответствующей коррекцией.

8.2. Выполнение сложения и вычитания с «длинными» данными

Очевидно, если длина данных превышает длину машинного слова (в нашем случае — байта), следует последовательно выполнять заданную операцию начиная с младших байтов, обеспечивая передачу переноса (заёма) от младших байтов к старшим.

8.2.1. Упакованный формат

Наиболее просто выполняются операции с **упакованными шестнадцатеричными числами**. В большинстве процессоров, наряду с командами *ADD* и *SUB*, предусмотрены команды *ADDC* и *SUBC*¹¹, которые складывают (вычитают) операнды с учётом значения флага *C*.

При отсутствии команд, автоматически учитывающих значение переноса (заёма) из предыдущего слова, необходимо обеспечить сохранность флага *C*, сформированного в текущем цикле операции, программно анализировать значение *C* в следующем цикле и при необходимости корректировать результат.

При работе с **упакованными десятичными числами** (код 8421) дополнительно требуется после операции над очередной парой байт выполнять *десятичную коррекцию* предварительного результата. Такая коррекция может выполняться специальными командами, описанными в разделе 2.3.1 настоящего пособия. При отсутствии таких команд в системе команд процессора эти же действия следует реализовать программно.

8.2.2. Распакованный формат

При наличии готовых процедур выполнения сложения/вычитания упакованных чисел можно упаковать распакованные данные, выполнить требуемую арифметическую операцию и, при необходимости, представить результат опять в распакованном формате. Однако, можно выполнять сложение/вычитание и непосредственно в распакованном формате.

При **сложении распакованных десятичных чисел** младшая тетрада байта предварительного результата нуждается в десятичной коррекции согласно выражению (1), стр. 17, а старшая тетрада результата всегда получается равной 'b0110' или 'b0111', учитывая, что старшие тетрады всех распакованных десятичных цифр равны 'b0011' и возможен перенос из младшей тетрады. Этот перенос должен быть передан в следующий разряд, однако флаг переноса *C* не будет сформирован. Фактически значение межразрядного переноса принимает младший бит старшей тетрады, поэтому результат после коррекции необходимо сохранить для анализа этого бита в следующем цикле сложения. После сохранения следует установить в старшей тетраде результата код 'b0011', приведя десятичную цифру результата к стандартному распакованному формату.

Вычитание распакованных десятичных чисел выполняется аналогично. Отличие – автоматическое формирование заёма в следующий разряд во флаге *C*.

¹¹В fN8 мнемоники этих команд – *ADC* и *SUBB*.

Арифметические операции над распакованными шестнадцатеричными числами (ASCII-кодами цифр 0..9 и $a..f$) связаны с несколькими дополнительными проблемами.

Во-первых, младшая тетрада ASCII-кодов $a..f$ принимает значения 'b0001' .. 'b0110' соответственно, поэтому перед началом арифметической операции необходимо добавить к ней 'b1001', причём такой коррекции подлежат только цифры $a..f$.

Во-вторых, старшие тетрады ASCII-кодов цифр 0..9 и $a..f$ отличаются друг от друга, причём одна из них нечётная ('b0011' для 0..9), а другая – чётная ('b0110' для $a..f$ ¹²). Поэтому не удаётся определить значение межразрядного переноса по одному из разрядов старшей тетрады результата и требуется сохранить значение AC для добавления к следующему разряду. Кстати, значение заёма из младшей тетрады правильно переходит во флаг C .

В-третьих, результат арифметической операции в старшей тетраде может принимать много различных значений, например, при сложении возможны значения старшей тетрады: 0x6, 0x7, 0x9, 0xA, 0xD. Однако, в результат надо поместить одно из двух значений старшей тетрады – в зависимости от получившегося значения младшей тетрады: 'b0011' для 0..9 или 'b0110' для $a..f$ ¹².

8.3. Задания

В ходе выполнения лабораторной работы необходимо разработать и отладить на ассемблере fN8 программу сложения (вычитания) многоразрядных десятичных (шестнадцатеричных) чисел произвольной разрядности, представленных как *целое без знака*. Варианты заданий (табл. 6) могут включать числа в упакованном формате в кодировке 8421 или в распакованном формате в кодировке ASCII. При этом возможен ввод операндов в заданном формате непосредственно в сегмент данных или с использованием контроллера клавиатуры. На рис. 22 – пример размещения упакованных десятичных чисел 9645012357 (по адресу 210) и 732581174 (по адресу 220) и результат сложения этих чисел по адресу 230.

	0	1	2	3	4	5	6	7	8	9
200	07	00	00	00	00	00	00	00	00	00
210	57	23	01	45	96	00	00	00	00	00
220	74	11	58	32	07	00	00	00	00	00
230	31	35	59	77	03	01	00	00	00	00
240	00	00	00	00	00	00	00	00	00	00

Рисунок 22. Десятичные упакованные операнды и результат их сложения

¹²Или 'b0100' для $A..F$

Если в задании предусмотрен вывод на дисплей, то как исходные операнды, так и результат должны размещаться в памяти в заданном формате и выводиться на дисплей. Рис. 23 иллюстрирует результаты операции вычитания десятичных чисел в распакованном формате, операнды вводятся с клавиатуры. И операнды и результат выводятся на дисплей.

	0	1	2	3	4	5	6	7	8	9	A
200	91	39	30	00	00	00	00	00	00	00	00
210	35	32	34	33	36	38	30	30	30	30	30
220	33	31	35	30	37	33	32	30	30	30	30
230	38	38	30	37	30	35	31	2D	30	30	30
240	00	00	00	00	00	00	00	00	00	00	00
250	00	00	00	00	00	00	00	00	00	00	00
260	00	00	00	00	00	00	00	00	00	00	00
270	32	33	37	30	35	31	33	00	00	00	00
280	00	00	00	00	00	00	00	00	00	00	00

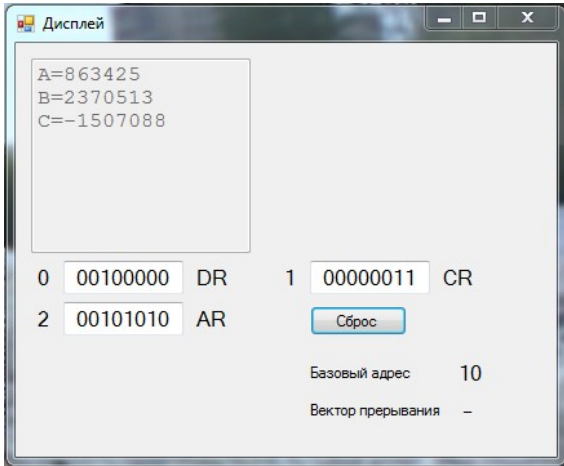


Рисунок 23. Десятичные распакованные операнды и результат вычитания

Варианты заданий приведены в табл. 6. Студент выбирает номер своего варианта в соответствии со своим порядковым номером **в списке группы**.

8.4. Содержание отчёта

Отчёт выполняется в электронном виде (формат .pdf или .doc/.docx) и должен содержать:

- 1) Формулировку задания.
- 2) Граф-схемы алгоритмов основной программы и вызываемых подпрограмм.
- 3) Текст программы (и подпрограмм) на языке Ассемблера fN8 (с комментариями!).
- 4) Распределение регистров и памяти сегмента данных.
- 5) Скриншот сегмента данных с исходными операндами и результатом.
- 6) Скриншот окна обозревателя дисплея с исходными операндами и результатом, если вывод на дисплей предусмотрен заданием.

Примечание. Программа и подпрограммы (пункт 3) должны быть представлены **не рисунком, а в текстовой форме**, позволяющем копировать его в окно компилятора fN8.

Таблица 6. Варианты заданий

№ вар.	Операция	Формат	Сист. числ.	Ввод	Вывод
1	Сложение	Упаков. (8421)	DEC	Из памяти	На дисплей
2	Вычитание	Упаков. (8421)	DEC	С клавиатуры	В память
3	Вычитание	Упаков. (8421)	HEX	С клавиатуры	На дисплей
4	Сложение	Упаков. (8421)	DEC	С клавиатуры	В память
5	Вычитание	Упаков. (8421)	HEX	Из памяти	На дисплей
6	Сравнение	Упаков. (8421)	DEC	С клавиатуры	На дисплей
7	Сложение	Упаков. (8421)	HEX	С клавиатуры	В память
8	Вычитание	Распак. (ASCII)	DEC	С клавиатуры	В память
9	Сравнение	Упаков. (8421)	DEC	Из памяти	На дисплей
10	Сложение	Упаков. (8421)	DEC	С клавиатуры	На дисплей
11	Вычитание	Упаков. (8421)	HEX	С клавиатуры	В память
12	Сложение	Распак. (ASCII)	DEC	Из памяти	В память
13	Сложение	Распак. (ASCII)	DEC	Из памяти	На дисплей
14	Вычитание	Упаков. (8421)	DEC	С клавиатуры	На дисплей
15	Вычитание	Упаков. (8421)	HEX	Из памяти	В память
16	Вычитание	Распак. (ASCII)	DEC	С клавиатуры	На дисплей
17	Сравнение	Распак. (ASCII)	DEC	Из памяти	На дисплей
18	Сравнение	Распак. (ASCII)	HEX	С клавиатуры	На дисплей
19	Вычитание	Упаков. (8421)	DEC	Из памяти	На дисплей
20	Сравнение	Упаков. (8421)	HEX	Из памяти	На дисплей
21	Сравнение	Распак. (ASCII)	DEC	С клавиатуры	На дисплей
22	Сложение	Упаков. (8421)	HEX	Из памяти	На дисплей
23	Сложение	Распак. (ASCII)	DEC	С клавиатуры	В память
24	Сравнение	Распак. (ASCII)	HEX	Из памяти	На дисплей
25	Вычитание	Распак. (ASCII)	DEC	Из памяти	В память
26	Вычитание	Распак. (ASCII)	DEC	Из памяти	На дисплей
27	Сложение	Упаков. (8421)	HEX	С клавиатуры	На дисплей
28	Сложение	Распак. (ASCII)	DEC	С клавиатуры	На дисплей
29	Сравнение	Упаков. (8421)	HEX	С клавиатуры	На дисплей
30	Сложение	Упаков. (8421)	DEC	Из памяти	В память
31	Сложение	Упаков. (8421)	HEX	Из памяти	В память
32	Вычитание	Упаков. (8421)	DEC	Из памяти	В память
33	Сложение	Распак. (ASCII)	HEX	Из памяти	На дисплей
34	Сложение	Распак. (ASCII)	HEX	С клавиатуры	В память
35	Сложение	Распак. (ASCII)	HEX	С клавиатуры	На дисплей
36	Сложение	Распак. (ASCII)	HEX	Из памяти	В память
37	Вычитание	Распак. (ASCII)	HEX	С клавиатуры	В память
38	Вычитание	Распак. (ASCII)	HEX	Из памяти	В память
39	Вычитание	Распак. (ASCII)	HEX	С клавиатуры	На дисплей
40	Вычитание	Распак. (ASCII)	HEX	Из памяти	На дисплей

9. Лабораторная работа № 2.2

Программирование систем контроля времени

Цель лабораторной работы – разработка на fN8 группы таймеров и/или секундомеров.

Требования к секундомеру:

- Точность измерения времени – 0,1 сек.
- Диапазон отсчёта времени – 0 .. 59 мин.: 59,9 сек.
- Формат отображения состояния секундомера: мм:сс,с
- Кнопки управления:

Пуск – запускает отсчёт времени с текущего состояния секундомера;

Stop – останавливается счёт, состояние секундомера не меняется;

Пауза – останавливается индикация, на экран выводится состояние секундомера на момент нажатия, счёт не останавливается. Повторное нажатие восстанавливает индикацию с текущего состояния секундомера;

Сброс – останавливается счёт и сбрасываются все разряды секундомера.

Требования к таймеру:

- Точность измерения времени – 0,1 сек.
- Диапазон задания интервала – 00мин:01сек .. 59мин:59сек
- Формат отображения состояния таймера: мм:сс[,с]
- Кнопки управления:

Пуск – запускает обратный отсчёт времени от текущего состояния таймера;

Сброс – останавливается счёт и сбрасываются все разряды таймера;

Цифровые клавиши – для ввода начального значения интервала в таймер.

9.1. Возможности таймерных систем микроконтроллеров и их программных моделей в fN8

В основе любой таймерной системы 8-разрядного микроконтроллера лежит двоичный 8- или 16-разрядный суммирующий (иногда – реверсивный) счётчик. На вход счётчика могут поступать по выбору счётные импульсы от одного из двух источников:

- 1) от внешнего вывода микроконтроллера; тогда таймерная система выступает в роли счётчика внешних событий. Счётные импульсы могут быть как периодическими, так и непериодическими и ограничены только верхним уровнем значения частоты;
- 2) от внутреннего источника системной тактовой частоты, обычно сниженной с помощью фиксированного или программируемого делителя. В этом случае таймерная система выполняет функции формирования или измерения временных интервалов.

В набор моделей внешних устройств fN8 входят две таймерные системы – Таймер-2 (раздел 3.4) и Таймер-5 (раздел 3.5), которые позволяют обеспечить отсчёт требуемого интервала времени.

Учитывая, что счётчики в таймерных системах микроконтроллеров работают обычно на сложение, очевидно, существует *два способа отсчёта заданного интервала времени*:

1. После очередного переполнения счётчика в него загружается константа, от которой ведётся отсчёт времени до верхнего предела счёта (переполнения). Загрузка константы может осуществляться программно или (если предусмотрен режим работы *Автозагрузка*) аппаратно, путём копирования константы из специального регистра.
2. *Сравнение (Сброс по совпадению)* – текущее значение счётчика сравнивается с константой, загруженной в специальный регистр; при совпадении значений счётчик аппаратно сбрасывается и отсчёт начинается с начала (от 0). Можно считать, что в специальном регистре задаётся произвольный верхний предел счёта.

Обнаружение факта завершения заданного интервала времени так же возможно одним из двух способов:

1. Программный анализ состояния флагов, отмечающих события *Переполнение* или *Совпадение*.
2. Обработка прерывания по соответствующему событию.

Рассмотрим возможности программных моделей таймеров в составе fN8 для формирования заданных временных интервалов и обнаружения факта их завершения.

- Таймер-2:

- В режиме 0 (*Простой счёт*) возможна программная загрузка произвольной константы, от которой будет вестись отсчёт, в счётчик TCNT. Верхний предел счёта в этом режиме всегда 0xFFFF, при переходе FFFF → 0000 устанавливается флаг переполнения OVF и формируется запрос на прерывание, если OVIE = 1.
- В режиме 3 (*Автозагрузка*) нужная константа будет загружаться в счётчик TCNT автоматически при его переполнении, если её предварительно поместить в регистр TIOR.
- В режиме 2 (*Сравнение*) в регистр TIOR помещается значение верхнего предела счёта¹³ по достижению которого устанавливается флаг OCF (сравнения) и формируется запрос на прерывание, если OCIE = 1.

- Таймер-5:

- В режиме 0 возможна, как и в Таймере-2, программная загрузка произвольной константы в TCNT, при этом верхний предел счёта задаётся выбором одной из трёх констант (0xFFFF, 0x0FFF, 0x00FF) или содержимым регистра ICR. При достижении верхнего предела счётчик TCNT сбрасывается в

¹³Для того, чтобы при TCNT = TIOR сбрасывался TCNT, необходимо установить флаг RT = 1 в регистре управления/состояния TSCR.

- 0, устанавливается флаг переполнения OVF и формируется запрос на прерывание, если OVIE = 1.
- Возможность автозагрузки в Таймере-5 не предусмотрены.
 - В режиме 1 (Сброс по совпадению) значение TCNT сбрасывается при TCNT = OCR, устанавливается флаг совпадения OCF и формируется запрос на прерывание, если ICIE = 1. При этом, значение OCR должно быть меньше выбранного верхнего предела счёта, иначе совпадение TCNT = OCR не наступит никогда.

9.2. Рекомендации по выполнению лабораторной работы

Шаг 1 – Распределение памяти

Для каждого из двух устройств следует выделить пять последовательных ячеек в сегменте данных для хранения значений десятков минут, минут, десятков секунд, секунд и десятых долей секунды. Рекомендуется располагать данные в ячейках таким образом, чтобы младшим элементам структуры данных соответствовали меньшие адреса, то есть в нашем случае в блоке из пяти соседних ячеек меньшему адресу соответствуют десятые доли секунды, а самому большому – десятки минут. Кроме двух пятиразрядных счётчиков в сегменте данных следует предусмотреть ячейки для хранения текущего состояния таймеров и секундомеров, флагов режимов и др.

Шаг 2 – Вывод на индикацию

Выбрав ячейки ОЗУ для счётчиков, попробуем вывести их содержимое на индикацию. Следует помнить, что кроме содержимого счётчика, следует выводить его имя, например C2 03.25.9 или T 00.18.0. В ячейках счётчика хранятся десятичные цифры, причём в разрядах десятков минут и десятков секунд – не более 5. Если в качестве устройства вывода задан контроллер 7-сегментной индикации и матричной клавиатуры, то с его помощью можно индицировать только одно, активное в данный момент, устройство. Под **активным** устройством здесь понимается не то устройство, которое сейчас запущено, а то из двух, которое мы выбрали активным (с помощью специальной кнопки).

Если информация выводится на символьный дисплей, то одновременно можно выводить состояния активного и неактивного устройств, но на действия органов управления должно реагировать только активное в данный момент устройство.

При выводе цифры на индикацию её следует преобразовать к формату вывода: для 7-сегментной индикации – в семисегментный код, для символьного дисплея – в ASCII-код. Можно хранить ASCII-код и непосредственно в разрядах счётчика (в старших тетрадах разрядов заменить 0000 на 0011) и выполнять с ним арифметические операции, только надо учитывать значение старшей тетрады при сравнении текущего значения разряда с его предельными значениями. А получение семисегментного кода цифры требует табличных преобразований.

Попробуйте загрузить в счётчик произвольные цифры, написать и отладить процедуру вывода его содержимого на заданное устройство вывода.

Шаг 3 – Органы управления

Для управления секундомером достаточно четырёх кнопок: *Пуск*, *Стоп*, *Сброс* и *Пауза*. Если в системе два секундомера, можно использовать эти кнопки только для активного в данный момент секундомера, нужно только добавить кнопку выбора активного.

Таймер управляется двумя кнопками: *Пуск* и *Сброс*, кроме того, необходимо обеспечить ввод интервала времени, который будет отсчитывать таймер. Для ввода 4-разрядного числа можно использовать цифровую клавиатуру или добавить две кнопки, независимо инкрементирующие значение минут и секунд. При этом следует учесть, что в разрядах десятков минут и десятков секунд не должны появляться цифры больше 5!

Роль кнопок могут исполнять произвольные клавиши клавиатуры компьютера или кнопки матричной клавиатуры блока *Клавиатура и индикация*. Событие *Нажатие клавиши* может контролироваться программой путём анализа значения флага готовности или с использованием механизма прерываний, причём выбрать способ может автор.

Разработайте граф-схему алгоритма, отображающую реакцию программы на нажатие используемых клавиш. Далее следует написать и отладить фрагмент программы, реализующей этот алгоритм.

Шаг 4 – Отсчёт времени

Способ формирования отрезка времени в 0,1 сек. целиком определяется пунктами задания В, С, и D. Значение констант, загружаемых в счётчик TCNT/регистр автозагрузки или в регистр сравнения, определяется расчётом или подбором. Следует помнить, что увеличение значения константы, загружаемой в счётчик TCNT или в регистр автозагрузки уменьшает отсчитываемый отрезок времени, а увеличение константы в регистре сравнения – увеличивает.

Событие *Переполнение/Сравнение* определяется в соответствии с пунктом D задания. При этом способ определения события *Нажата клавиша* не регламентирован. С учётом пункта D задания и отсутствия во всех вариантах задания необходимости программировать динамическую индикацию, основной цикл программы может быть реализован одним из следующих вариантов:

- последовательный анализ флагов *Переполнение/Сравнение* счётчика и *Нажата клавиша* клавиатуры с вызовом соответствующих подпрограмм;
- анализ флага *Переполнение/Сравнение* и переход на обработчик прерывания по событию *Нажата клавиша*;
- анализ флага *Нажата клавиша* и переход на обработчик прерывания по событию *Переполнение/Сравнение*;
- пустой цикл с переходом на обработчики прерываний по соответствующим событиям.

Шаг 5 – Программы INC и DEC

Для реализации функции секундомера необходимо разработать подпрограмму под

условным названием INC¹⁴, которая к пятиразрядной структуре счётчика секундомера добавляет единицу с учётом того, что первый (десять доли секунды), второй (секунды) и четвёртый (минуты) разряды принимают значения в диапазоне 0...9, а третий и пятый (десятки секунд и минут) – в диапазоне 0...5.

Для функционирования таймера требуется подпрограмма, вычитающая единицу из такой же пятиразрядной структуры таймера с теми же ограничениями на значения разрядов.

9.3. Содержание отчёта

Отчёт выполняется в электронном виде (формат .pdf или .doc/.docx) и должен содержать:

- 1) Формулировку задания.
- 2) Граф-схемы алгоритмов основной программы и вызываемых подпрограмм.
- 3) Текст программы (и подпрограмм) на языке Ассемблера fN8 (с комментариями!).
- 4) Распределение регистров и памяти сегмента данных.
- 5) Несколько скриншотов устройства вывода в разных режима работы секундомеров (таймеров).

Примечание. Программа и подпрограммы (пункт 3) должны быть представлены **не рисунком, а в текстовой форме**, позволяющем копировать его в окно компилятора fN8.

9.4. Контрольные вопросы

- 1) Какие режимы предусмотрены в работе Таймера-2 (Таймера-5)? Какие из них можно использовать для решения задач, поставленных в Вашем индивидуальном задании? Как?
- 2) Как следует записывать (считывать) информацию в (из) 16-разрядный регистр таймера с помощью команд, оперирующих байтами?
- 3) Какими способами можно задать интервал времени, в течение которого таймер достигнет своего верхнего предела?
- 4) Как обнаружить факт достижения таймером значения верхнего предела?
- 5) Как использовать подсистему прерываний для обнаружения факта нажатия клавиши? Можно ли обнаружить этот факт без использования подсистемы прерываний?
- 6) Как определить код нажатой клавиши?
- 7) Как использовать подсистему прерываний для обнаружения факта завершения заданного для системного таймера интервала времени?
- 8) Как сформировать таблицу векторов прерываний при использовании подсистемы прерываний для обнаружения фактов нажатия клавиши и завершения заданного для системного таймера интервала времени?

¹⁴Рекомендуется придумать подпрограммам другие имена, не совпадающие с мнемониками команд процессора.

9.5. Варианты заданий

Задание определяется следующими параметрами:

А) Система:

- 1) 2 таймера
- 2) 2 секундомера
- 3) 1 секундомер и 1 таймер

В) Счётчик времени:

- 1) Таймер_2
- 2) Таймер_5

С) Способ формирования временного интервала:

- 1) Сброс по совпадению
- 2) Автозагрузка или программная загрузка начального значения счёта

Д) Способ обнаружения переполнения/сравнения счётчика времени TCNT:

- 1) Анализ соответствующего флага
- 2) По прерыванию

Е) Устройство ввода:

- 1) Контроллер клавиатуры
- 2) Матричная клавиатура

Ф) Устройство вывода:

- 1) Символьный дисплей
- 2) 7-сегментный индикатор

Таблица 7. Варианты заданий

№ вар.	Задание	№ вар.	Задание	№ вар.	Задание
1	A1-B2-C1-D1-E2-F2	9	A2-B2-C2-D1-E2-F2	17	A3-B1-C2-D1-E1-F2
2	A1-B1-C1-D1-E2-F2	10	A3-B1-C2-D1-E1-F1	18	A2-B2-C2-D1-E1-F1
3	A2-B2-C1-D1-E2-F2	11	A1-B2-C2-D1-E2-F2	19	A3-B1-C2-D1-E2-F2
4	A2-B2-C1-D2-E2-F2	12	A3-B1-C2-D1-E2-F2	20	A1-B2-C1-D2-E2-F2
5	A3-B2-C2-D1-E1-F2	13	A1-B2-C1-D1-E1-F1	21	A2-B2-C2-D2-E1-F2
6	A2-B1-C2-D2-E2-F2	14	A3-B2-C2-D1-E1-F1	22	A1-B2-C2-D2-E2-F2
7	A1-B1-C2-D1-E1-F2	15	A3-B1-C1-D1-E1-F1	23	A2-B1-C2-D1-E2-F2
8	A2-B2-C2-D1-E1-F1	16	A3-B1-C2-D2-E1-F2	24	A1-B2-C1-D1-E2-F2

Список литературы

- [1] Жмакин А. П. Архитектура ЭВМ : 2-е изд., - СПб.: БХВ-Петербург, 2010.
- [2] Основы функционирования ЭВМ: методические указания к выполнению цикла лабораторных работ / сост. А. П. Жмакин; Курск. гос. ун-т. – Курск, 2017.

Составитель

Анатолий Петрович Жмакин

Основы функционирования ЭВМ

**Методические указания к выполнению цикла лабораторных работ
(Выпуск 2)**

Курский государственный университет
305000, г. Курск, ул. Радищева, 33

Лицензия № 06248 от 12.11.2001 г.

Подписано в печать .
Формат 60 × 84/16. Печать офсетная.
Усл.печ.л. 4,3. Тираж 100 экз.
Заказ №

Отпечатано в отделе информационно-методического обеспечения КГУ